Prioritized Elastic Round Robin: An Efficient and Low-Latency Packet Scheduler with Improved Fairness

Salil S. Kanhere
and
Harish Sethu

Technical Report DU-CS-03-03
Department of Computer Science
Drexel University
Philadelphia, PA 19104
July 2003

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
JUL 2003		**00-07-2003 to 00-07-2003**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Prioritized Elastic Round Robin: An Efficient and Low-Latency Packet Scheduler with Improved Fairness	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Department of Computer Science,Drexel University,Philadelphia,PA,19104**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**43**	

Prioritized Elastic Round Robin: An Efficient and Low-Latency Packet Scheduler with Improved Fairness [1]

Salil S. Kanhere and Harish Sethu

Abstract

In emerging high-speed integrated-services packet-switched networks, fair packet scheduling algorithms in switches and routers will play a critical role in providing the Quality-of-Service (QoS) guarantees required by real-time applications. Elastic Round Robin (ERR), a recently proposed scheduling discipline, is very efficient with an O(1) work complexity. In addition, it has superior fairness and delay characteristics in comparison to other algorithms of equivalent efficiency. However, since ERR is inherently a round robin scheduling algorithm, it suffers from the limitations of all round robin schedulers such as (i) bursty transmission and (ii) the inability of the flows lagging in service to receive precedence over the flows that have received excess service. Recently, Tsao and Lin have proposed a new scheme, Pre-order Deficit Round Robin, which tries to eliminate the problems associated with the round robin service order of Deficit Round Robin (DRR). In this report, we present a new scheduling discipline called Prioritized Elastic Round Robin (PERR), based on a similar principle as Pre-order DRR but in a modified and improved form, which overcomes the limitations of ERR. We derive an upper bound on the latency achieved by PERR using a novel technique based on interpreting the scheduling algorithm as an instance of a nested version of ERR. Our analytical results show that PERR has better fairness characteristics and a significantly lower latency bound in comparison to other scheduling disciplines of equivalent work complexity such as DRR, ERR and Pre-order DRR. We further present simulation results, using both synthetic and real traffic traces, which illustrate the improved performance characteristics of PERR.

Key words: Fair queueing, Elastic Round Robin, low latency, relative fairness bound, quality of service, Gini index.

Email addresses: salil@ece.drexel.edu (Salil S. Kanhere), sethu@ece.drexel.edu (Harish Sethu).

[1] This work was supported in part by NSF CAREER Award CCR-9984161 and U.S. Air Force Contract F30602-00-2-0501.

1 Introduction

The Internet traffic today is increasingly dominated by new real-time multimedia applications such as audio/video-on-demand, IP telephony, and multimedia teleconferencing, all of which prefer some guarantee of an acceptable quality of service (QoS) in the transmission and delivery of data. These different applications have varying traffic characteristics with different requirements, and rely on the ability of the network to provide QoS guarantees with respect to several quality measures, such as end-to-end delay, bandwidth allocation, delay jitter, and packet loss. This requires a scalable QoS mechanism to efficiently apportion, allocate and manage limited network resources among competing users. Fair, efficient and low-latency packet scheduling algorithms used at the output links of switches and routers form an important component of such a mechanism [1]. Fair scheduling becomes especially critical in access networks, within metropolitan area networks and in wireless networks where the resource capacity constraints tend to be significantly limiting to high-bandwidth multimedia applications today. Even with the over-provisioning of resources such as is typical in the Internet core, fairness in scheduling is essential to protect flows from other misbehaving flows triggered by deliberate misuse or malfunctioning software on routers or end-systems. Fairness in the management of resources is also helpful in countering certain kinds of denial-of-service attacks. Fair schedulers have now found widespread implementation in switches and Internet routers [2, 3].

Given multiple packets awaiting transmission through an output link, the function of a packet scheduler is to determine the exact sequence in which these packets will be transmitted through the link. Desirable properties of a packet scheduler include:

- *Fairness in bandwidth allocation:* This characteristic of a scheduler is most frequently judged based on the max-min notion of fairness [1, 4].
- *Low latency:* An appropriate measure of packet schedulers in this regard, especially for schedulers seeking to provide guaranteed services, is the upper bound on the length of time it takes a new flow to begin receiving service at the guaranteed rate [5].
- *Low implementation complexity:* A per-packet work complexity of $O(1)$ with respect to the number of flows is considered most desirable for ease of implementation, especially in hardware switches.

Often, there is a conflict between a simple and efficient implementation on the one hand and low latency and fairness on the other. In this report, we present a new scheduling discipline called *Prioritized Elastic Round Robin (PERR)* with a work complexity of $O(1)$ with respect to the number of flows and with better fairness and latency characteristics than other known schedulers of equivalent complexity. In addition, these characteristics of PERR are comparable to those of significantly more complex schedulers such as WFQ. In this report, we present analytical proofs

of these characteristics of PERR. Further, using real gateway traffic traces, and based on a new measure of fairness borrowed from the field of economics, we present simulation results that demonstrate the improved fairness characteristics of PERR.

1.1 Related Work and Motivation

Fair schedulers—those discussed in the research literature as well as those currently implemented in switches and routers—are often based on the *max-min* notion of fairness [1, 4], a strategy for apportioning a shared resource among multiple requesting entities. The central idea of the max-min strategy is that no entity receives a share of the resource larger than its demand and that entities with unsatisfied demands receive equal shares of the resource. The Generalized Processor Sharing (GPS) [6] is an ideal but unimplementable scheduler that assumes fluid flow traffic to exactly satisfy the above classical notion of fairness.

Over the last several years, a large number of different packet-based scheduling algorithms have been proposed with an aim to approximate the GPS scheduler. These algorithms can be broadly classified into the following two categories—sorted-priority schedulers and frame-based schedulers. Sorted-priority schedulers maintain a global variable known as the virtual time or the system potential function. A timestamp corresponding to each packet is computed based on this variable, and packets are scheduled in increasing order of their timestamps. Examples of sorted-priority schedulers are Weighted Fair Queueing (WFQ) [6, 7], Self-Clocked Fair Queueing (SCFQ) [8], Start-Time Fair Queueing (SFQ) [9], Frame-based Fair Queueing [2] (FFQ) [10] and Worst-Case Fair Weighted Fair Queueing (WF^{2}Q) [11]. The sorted-priority schedulers primarily differ in the manner in which they compute the global virtual time function. They generally provide good fairness and a low latency bound but are not very efficient due to the complexity involved in computing the virtual time function and in having to maintain a sorted list of packets based on their timestamps. WFQ and WF^{2}Q suffer an $O(n)$ time complexity to compute the virtual time function, where n is the number of flows sharing the same output link. Schedulers such as SCFQ, SFQ and FFQ reduce the complexity of computing the virtual time to $O(1)$ by using approximations of the correct virtual times. However, they still need to maintain a sorted list of packets based on their timestamps incurring a per-packet work complexity of $O(\log n)$.

In frame-based schemes, on the other hand, the scheduler provides service opportunities to the backlogged flows in a particular order and, during each service op-

[2] Note that Frame-based Fair Queueing, in spite of its name, is actually a sorted-priority scheduling discipline. The algorithm uses a framing approach similar to that used in frame-based schedulers to update the state of the system. However, as in sorted-priority schedulers, packets are transmitted based on their timestamps.

portunity, the intent is to provide the flow an amount of service proportional to its fair share of the bandwidth. Examples of such schedulers are Deficit Round Robin (DRR) [12], Surplus Round Robin (SRR) [13–15] and Elastic Round Robin (ERR) [16]. The frame-based schedulers do not maintain a global virtual time function and also do not require any sorting among the packets available for transmission. This reduces the implementation complexity of frame-based scheduling disciplines to $O(1)$, making them attractive for implementation in routers and, especially so, in hardware switches. However, these frame-based schedulers suffer the following disadvantages:

- *High Start-Up Latency:* The above frame-based schedulers operate in a round-robin fashion, with each active flow receiving exactly one opportunity to transmit in each round. When a new flow becomes active, it has to wait until all other previously active flows receive their service opportunity before it can receive service from the scheduler. With large numbers of flows, this time period can be very large, especially in comparison to sorted-priority schedulers such as WFQ and WF^2Q.
- *Bursty output:* Each flow is served over a continuous time interval during its round robin service opportunity leading to a bursty packet stream at the output of the scheduler for any given flow. This is not an ideal situation for real-time multimedia traffic since even smooth flows are rendered bursty as they exit the scheduler.
- *Delayed correction of unfairness:* If a flow receives very little service in a particular round, it is compensated with proportionately more service in the next round. While this disadvantaged flow waits for its compensation in the next round, other flows which have already received more service than their fair share in the previous round continue to receive yet more service before the disadvantaged flow receives its opportunity.
- *Compounded Jitter:* When a flow's arrival pattern at the scheduler has high jitter, it can frequently happen that the flow runs out of packets even before it has received its fair share of service during its service opportunity. At this point, the scheduler moves on to serve other currently active flows in a round-robin fashion. Our flow with high jitter will receive its next opportunity only in the next round after all the other active flows have completed their transmissions. This further increases the jitter in the output of the scheduler since a delayed packet that just misses its service opportunity in a certain round ends up experiencing significant additional delay because of having to wait for all the other active flows to complete their transmissions.

These weaknesses of frame-based schedulers discussed above are caused by the same features that are common to these schedulers:

(1) The round robin nature of the service order.
(2) Each flow receives its entire share of service in the round at once in one service opportunity.

Overcoming these weaknesses while preserving the $O(1)$ complexity of frame-based schedulers forms the primary motivation behind this report.

At least a few proposals have been made in the last few years to overcome the limitations of frame-based schedulers discussed above. The Nested-DRR scheduler proposed in [17] tries to eliminate some of these limitations of the DRR scheduler [12]. For each flow i, the DRR scheduler maintains a *quantum*, Q_i, which represents the ideal service that the flow should receive in each round of service. If the entire quantum is not used in a given round, the deficit is recorded and used to compensate the flow in the next round. Nested-DRR splits each DRR round, referred to as an *outer round*, into one or more smaller *inner rounds* and executes a modified version of the DRR algorithm within each of these inner rounds. If Q_{min} is the quantum assigned to the flow with the lowest reserved rate, the Nested-DRR scheduler tries to serve Q_{min} worth of data from each flow during each inner round. During an outer round, a flow is considered to be eligible for service in as many inner rounds as are required by the scheduler to exhaust its quantum. The Nested-DRR scheduler, just like the DRR scheduler, has a per-packet work complexity of $O(1)$ as long as the largest packet that may potentially arrive in flow i is smaller than Q_i.

The technique of nesting smaller rounds within a round as in the Nested-DRR scheduler may be adapted for use with other $O(1)$ schedulers such as ERR and SRR. The Nested-DRR scheduler results in a significant improvement in the latency in comparison to DRR, but only in those cases in which there is a significant difference between the quanta assigned to the flows. If all flows are of the same weight, the behavior of the Nested-DRR scheduler is identical to that of DRR. Further, the improvement gained in fairness or latency is again limited by the fact that, within each inner round, the nested scheduler still serves the active flows in a round robin manner.

The Pre-order DRR algorithm proposed in [18] combines the nesting technique explained above with a scaled down version of the sorting of packets used in the sorted-priority schedulers and thus, succeeds in overcoming some of the drawbacks of the DRR scheduler. The Pre-order DRR scheduler adds a limited number of priority queues in which packets wait before being actually scheduled for transmission. The packets that are transmitted in a DRR round from each flow are now classified into these queues depending on the percentage of the flow's quantum that each packet will utilize following its transmission. Thus, the transmission sequence of the packets in a round in DRR is *reordered* allowing certain packets to receive priority over others, resulting in an improvement in the latency and fairness properties.

In this report, we propose a novel packet scheduler, *Prioritized Elastic Round Robin (PERR)*, which exhibits improved fairness and latency characteristics in comparison to other known schedulers of equivalent complexity, including Pre-order DRR discussed earlier. The total service received by a flow in a round in PERR is identical to the service received by the flow in the corresponding round in ERR. However, in PERR, this service received by a flow is split into several parts over the course of the round. The PERR scheduler, borrowing the principle used in Pre-order DRR, re-orders the transmission sequence within each round of the ERR scheduler. The transmission sequence of the packets in a round is reordered to allow the flows that have received less service in the previous round to gain precedence over the other flows in the current round. The exact manner in which the transmission sequence is re-ordered depends on a certain per-flow state that indicates how far ahead or behind a flow is in consuming its share of bandwidth. As in the Pre-order DRR, the scheduler maintains a limited number, p, of priority queues which serve to implement the re-ordered transmission sequence.

The PERR scheduler achieves a significant improvement in the fairness, latency, and delay jitter characteristics and addresses several of the weaknesses (such as burstiness of output traffic) of round-robin schedulers. In addition to its superior fairness and latency characteristics, the PERR scheduler holds several advantages over other schedulers, such as Pre-order DRR, that have attempted to address these weakness of round-robin schedulers. For example, at the start of a round, the Pre-order DRR scheduler has to classify all the packets that will be transmitted by the active flows in that round into the priority queues prior to the beginning of the transmission of the packets. On the contrary, the PERR scheduler simply has to classify the flows (as opposed to packets) present in the *ActiveList* into its priority queues before the start of the round. This reduces the buffering requirements and the delay through the finite state machines managing the transmission scheduling, since classifying all the packets in the round into priority queues requires considerably more time than simply sorting the flow identifiers. In addition, in comparison to Pre-order DRR, this allows a more dynamic re-ordering of the transmission sequence based on the latest state of the flows, leading to improved fairness at all instants of time.

As shown in [16, 19], the ERR scheduler has a couple of important advantages in comparison to DRR. Since PERR is based on the ERR scheduler, PERR inherits some of these advantages as well. For example, unlike DRR or Pre-order DRR, the PERR scheduler does not require the knowledge of the transmission time of each packet prior to the scheduling operation. As a result, the scheduler can be used in other networks such as wormhole networks, where the transmission time of a packet depends not only on the size of the packet but also the downstream congestion. For the same reasons, PERR—but not DRR or Pre-order DRR—may be used in ATM

networks transmitting IP packets over AAL5, where the end of the packet is not known until the arrival of the last ATM cell corresponding to the packet.

Further, in this report, we analytically prove the fairness and latency properties of PERR, using a novel approach based on interpreting the PERR scheduler as an instance of the Nested Deficit Round Robin (Nested-DRR) discipline discussed in [17]. We prove that the latency bound obtained in this report using this approach is tight. We also show that the per-packet work complexity of the PERR scheduler is $O(1)$ with respect to the number of flows and $O(\log p)$ with respect to the number of priority queues, p. The fairness and latency measures used in this report and in other literature on scheduling algorithms, however, are only bounds and do not accurately capture the behavior of the scheduler most of the time under normal circumstances. Therefore, we borrow a measure of inequality used in the field of economics to comparatively judge the instantaneous fairness achieved by PERR with both synthetic traffic and real gateway traffic traces.

1.3 Organization

The rest of the report is organized as follows. Section 2 presents an overview of the ERR algorithm and illustrates some of its disadvantages. In Section 3, we present our new scheme, Prioritized Elastic Round Robin (PERR), which aims at overcoming the drawbacks of ERR. Section 4 discusses the interpretation of PERR bandwidth allocations as an instance of allocations in a nested version of ERR. Section 5 presents analytical results which prove the fairness, latency and efficiency properties of PERR. Section 6 provides a tabulated summary of PERR in comparison to other guaranteed-rate scheduling disciplines. Section 7 presents simulation results using both synthetic traffic and real gateway traffic traces. Finally, Section 8 concludes the report.

2 Elastic Round Robin (ERR)

Elastic Round Robin (ERR) [16,19] is a recently proposed fair and efficient scheduling discipline for best-effort traffic as well as traffic that requires guaranteed services. In this section we present a brief overview of ERR, upon which the PERR scheduler is based.

Consider an output link of transmission rate r, access to which is controlled by the ERR scheduler. Let n be the total number of flows and let ρ_i be the reserved rate for flow i. Let ρ_{min} be the lowest of these reserved rates. Note that since all the flows share the same output link, a necessary constraint is that the sum of the reserved rates be no more that the transmission rate of the output link. In order that each

flow receives service proportional to its guaranteed rate, the ERR scheduler assigns a weight to each flow. The weight assigned to flow i, w_i is given by,

$$w_i = \frac{\rho_i}{\rho_{min}} \tag{1}$$

Note that for any flow i, $w_i \geq 1$.

A flow is said to be *active* during an interval of time if it always has packets awaiting service during this interval. A maximal interval of time during which a flow is continuously active is called an *active period* of the flow. The ERR scheduler maintains a linked list of the active flows, called the *ActiveList*. At the start of an active period of a flow, the flow is added to the tail of the *ActiveList*. A *round* is defined as one round robin iteration during which the ERR scheduler serves all the flows that are present in the *ActiveList* at the outset of the round. Each flow receives no more than one service opportunity during each round. The scheduler selects the flow at the head of the *ActiveList* for service and calculates its *Allowance*, defined as the number of bits that the flow can transmit during the current round. This allowance, however is not a rigid one and is actually *elastic*. In other words, a flow may be allowed to receive more service in the current round than its allowance. Let $A_i(s)$ represent the allowance for flow i in some round s and let $Sent_i(s)$ represent the actual service received by flow i during this round. The ERR scheduler will begin the transmission of the next packet in the queue of flow i if and only if $Sent_i(s)$ is less than $A_i(s)$. Note that the last packet transmitted by flow i during round s may cause $Sent_i(s)$ to exceed $A_i(s)$. In this case, flow i receives more than its fair share of the bandwidth. This excess usage is recorded in the *Surplus Count (SC)*. Let $SC_i(s)$ represent the surplus count of flow i in round s. Following the service of flow i during the s-th round, its surplus count is computed as follows:

$$SC_i(s) = Sent_i(s) - A_i(s) \tag{2}$$

Let $MaxSC(s)$ denote the largest normalized surplus count among all flows served during round s. In other words,

$$MaxSC(s) = \max_{\forall i \text{ served in round } s} \left\{ \frac{SC_i(s-1)}{w_i} \right\} \tag{3}$$

This quantity is recursively used to compute the allowance of each flow in the subsequent round, as follows:

$$A_i(s) = w_i(1 + MaxSC(s-1)) - SC_i(s-1) \tag{4}$$

Note that the allowance of a flow in a certain round depends on the surplus count of the other flows in the previous round. Each flow seeks to catch up with the flow that has the largest normalized surplus count among all flows served in the previous round.

Let m be the size in bits of the largest packet that is actually served during the execution of a scheduling algorithm. It is proved in [16] that for any flow i and round s,

$$0 \leq SC_i(s) \ \leq \ m-1 \tag{5}$$
$$0 \leq MaxSC(s) \ \leq \ m-1 \tag{6}$$

The above results will be important in our later analysis of PERR in Section 5.

3 Prioritized Elastic Round Robin

The basic principle of the PERR scheduler involves modifying the transmission sequence of the packets that are scheduled within each round in ERR. This re-ordering is performed upon the transmission of each packet, and is carried out based on the amount of each active flow's allowance for the round that is actually consumed until the instant that the re-ordering is executed. This allows each flow to utilize its allowance in pieces over the duration of each round. The reordering is implemented through the use of priority queues, which are nothing but linked lists of flow identifiers. The scheduler transmits packets from the flows in the highest priority queue first, and begins serving a flow in another priority queue only after all higher priority queues are empty. The core aspect of the PERR algorithm is how it manages these priority queues and rearranges flows amongst these priority queues.

In this section, we present a detailed description of the PERR algorithm. We begin our discussion by introducing certain important definitions that are essential to understanding the rationale behind the design of the PERR scheduler.

As in ERR, let $Sent_i(s)$ represent the total service received by flow i in the s-th round of service. Assume that a total of y packets are transmitted from flow i in round s. The packets are labeled as 1, 2, ..., y, indicating their position in the transmission sequence of flow i. Let $Sent_i^k(s)$ represent the total data transmitted by flow i after completion of the transmission of the first k packets of the flow during the s-th round. Note that the service received by flow i in round s prior to the transmission of its first packet in that round is equal to zero, i.e., $Sent_i^0(s) = 0$. Also, note that $Sent_i^y(s) = Sent_i(s)$, since both represent the total service received by flow i in round s. In general, of course,

$$0 \leq Sent_i^k(s) \leq Sent_i(s), \ \ 0 \leq k \leq y$$

The following defines a quantity that tracks the unused portion of a flow's allowance, and thus serves to help in determining the priority queue into which the

flow should be placed.

Definition 1 *Define the* Unserved Allowance *of a flow at any given instant of time during a certain round as the flow's allowance for the round minus the amount of traffic transmitted by the flow during the round until that instant.*

Let $UA_i^k(s)$ represent the unserved allowance of flow i after the transmission of its k-th packet during the s-th round. In general, $UA_i^k(s)$ is computed as follows:

$$UA_i^k(s) = A_i(s) - Sent_i^k(s) \tag{7}$$

At the start of round s, before service for flow i begins, $UA_i^0(s)$ is exactly equal to the flow's allowance for the round, $A_i(s)$. Note that the last packet transmitted from flow i in round s may cause the flow to exceed its allowance. This may result in a negative value of $UA_i^k(s)$.

Definition 2 *Define $UA_i^{max}(s)$ as the maximum possible value of the unserved allowance of flow i in round s. At the start of each round, the unserved allowance of a flow is initialized to its allowance for the round, as defined in Equation (4). Therefore, $UA_i^{max}(s)$ equals the maximum possible value of the right-hand side of Equation (4). Using Equation (5), we have,*

$$UA_i^{max}(s) = w_i(1 + MaxSC(s-1)) \tag{8}$$

The ratio of the unserved allowance of a flow at a given instant and the allowance of the flow for the entire round represents the fraction of the allowance of a flow that is not yet consumed until the given instant. This ratio accurately captures how far ahead or behind a flow is in comparison to other flows in obtaining its fair share of service, and may therefore be used in placing flows in specific priority queues. However, an approximation to this quantity is necessary to ensure a per-packet work complexity of $O(1)$ for PERR.

Normalizing the unserved allowance of a flow with respect to its maximum possible value (instead of the actual allowance of the flow for the round) represents one measure, though not necessarily the most accurate measure, of the fraction of its allowance that is not yet consumed. The PERR scheduler uses this approximation which is necessary for the efficient implementation of the scheduler. It will be shown in later sections of this report that, in spite of this approximation, the PERR scheduler achieves better fairness than other known $O(1)$ schedulers.

Definition 3 *The* Unserved Allowance Quotient *of a flow at any given instant is defined as the ratio of the unserved allowance of the flow at that instant and its maximum possible unserved allowance, $UA_i^{max}(s)$. Let $Q_i^k(s)$ represent the unserved allowance quotient of flow i after the transmission of the k-th packet of flow i during the s-th round.*

$Q_i^k(s)$ is given by,

$$Q_i^k(s) = \frac{UA_i^k(s)}{UA_i^{max}(s)} = \frac{UA_i^k(s)}{w_i(1 + MaxSC(s-1))} \tag{9}$$

For purposes of brevity, in the rest of this report, the unserved allowance quotient will be simply referred to as the *quotient*.

The quotient of a flow at any given instant during a round represents the approximate fraction of its unserved allowance that can be used by the flow in the remainder of the round. The ERR scheduler never begins dequeuing the new packet from a flow i if $Sent_i^k(s)$ is equal to or more than the allowance, $A_i(s)$. Thus, the next packet of a flow is transmitted in the same round as the previous packet if and only if $UA_i^k(s)$ is positive. This in turn implies that a flow i, after the transmission of its k-th packet in round s, is eligible for more service in the same round if and only if,

$$0 < Q_i^k(s), \;\; 0 \leq k \leq y \tag{10}$$

where y is the number of packets of flow i served during round s.

The quotient for flow i at the start of round s is equal to $Q_i^0(s)$. Using Equations (4), (7), (8) and (9) we have,

$$Q_i^0(s) = \frac{UA_i^{max}(s) - SC_i(s-1)}{UA_i^{max}(s)}$$

Simplifying further, we get,

$$SC_i(s-1) = (1 - Q_i^0(s))\, UA_i^{max}(s) \tag{11}$$

This indicates that flow i has already used up $(1-Q_i^0(s))$-th fraction of its $UA_i^{max}(s)$ in the excess service that it received in the previous round $(s-1)$. If the quotient for a flow at the start of a round is equal to unity, it implies that the surplus count of the flow following its service in the previous round is zero, i.e., the flow did not receive any excess service in the previous round.

In general, the larger the quotient of a flow, the lesser the proportion of its *UnservedAllowance* that has been expended in the current round.

Definition 4 *Define $Q^{max}(s)$ as the maximum of the quotients among all active flows at the start of round s.*

Since the *Unserved Allowance* for each flow at the start of a round is equal to its allowance, using Equations (4), (8) and (9), we have,

$$Q_i^0(s) = \frac{w_i(1 + MaxSC(s-1)) - SC_i(s-1)}{w_i(1 + MaxSC(s-1))}$$

This implies that the flow i with the least value of $\frac{SC_i(s-1)}{w_i}$, which is the normalized surplus count at the end of the previous round $(s-1)$, will be the one with the maximum value of the quotient among all active flows at the start of round s.

Ideally the scheduler should serve a packet from the flow with the largest quotient among all the active flows since it has received the least service in the current round. However, the complexity of maintaining a sorted list of active flows based on their quotients, and the complexity of computing the maximum in this list prior to each packet transmission is high. Given n flows, the work complexity of the scheduler prior to each packet transmission would be $O(\log n)$. The PERR scheduler avoids this by grouping the flows into a limited number of priority queues.

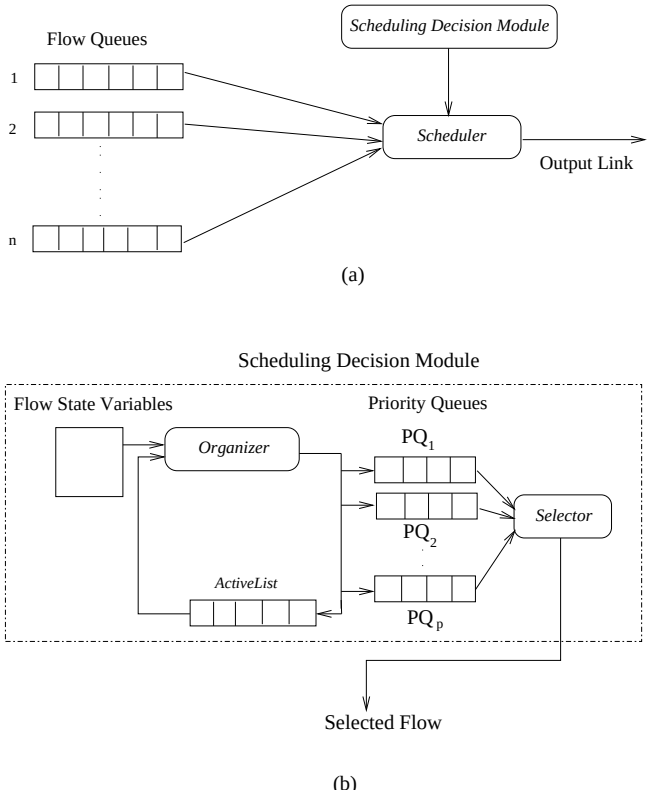

Fig. 1. Block Diagram of (a) PERR Scheduler and (b) Scheduling Decision Module of PERR

Fig. 1(a) illustrates a block diagram of a generic scheduler. The *Scheduling Decision Module* which determines the order in which packets are served from the flow queues is the heart of the scheduler. Fig. 1(b) details the architecture of the *Scheduling Decision Module* of the PERR scheduler which is responsible for selecting the next flow for service. As can be seen from Fig. 1(b), an *Organizer*, p priority queues and a *Selector* are appended to the original *Scheduling Decision Module* of ERR. Let PQ_1, PQ_2, ... and PQ_p denote the priority queues in the descending order of priority with PQ_1 representing the queue with the highest priority. Unlike the priority queues in Pre-order DRR which have to buffer the packets that will be transmitted in the round in progress, these queues in PERR simply contain the flow

identifiers. As in ERR, the PERR scheduler maintains a linked list, called the *ActiveList*, of flows which are active. However, the flows in the *ActiveList* are not served in a round robin manner as in ERR. This is a list of the active flows that have exhausted their allowance in the current round but, will be eligible for receiving service in the subsequent round. It is the task of the *Organizer* to determine the order in which the flows will receive service in a round. At the start of the round, the *Organizer* module classifies the active flows present in the *ActiveList* into several classes according to their *Unserved Allowance Quotient* and places them into the corresponding priority queue. Since there are p priority queues, the *Organizer* can classify the flows into p classes. In general, class z of flow i after serving the k-th packet of the s-th round is derived as,

$$z = (p+1) - \left\lceil p \times \frac{Q_i^k(s)}{Q^{max}(s)} \right\rceil \tag{12}$$

Note that at the start of the s-th round, $k = 0$ for all the flows in the above equation.

Note that the quotient of a flow is a monotonically decreasing function of time over the duration of a round. Using this fact and the definition of $Q^{max}(s)$, we can conclude that during round s, the quotients of all the active flows will always be less than or equal to $Q^{max}(s)$, and thus z is always a non-negative quantity. By the above method, the flow with the maximum quotient at the start of the round s is initially added into the highest priority queue, PQ_1.

Note that $Q^{max}(s)$ is computed only at the start of the round and need not be updated as the round progresses. The computation of $Q^{max}(s)$ simply requires the scheduler to record the least value of the normalized surplus count amongst all the flows in the *ActiveList*. This can be easily accomplished in $O(1)$ time by carrying out a simple comparison operation as the flows are added into the *ActiveList* after exhausting their allowance in the previous round.

Consider a situation where a flow i becomes active for the first time while some round s is in progress. It may be possible that the the initial value of the quotient of flow i, $Q_i^0(s)$ is greater than $Q^{max}(s)$. Using Equation (12), it is seen that the priority class for flow i is less than 1, the highest priority class. One possible solution would be to delay the service of flow i until the next round as is done in ERR. However, this would result in an increased latency for flow i since it would then have to wait until all the active flows have exhausted their allowance before receiving any service. The PERR scheduler instead simply adds flow i into the highest priority queue PQ_1. This eliminates the increased latency that would be otherwise experienced by flow i.

When the scheduler is ready to transmit, the *Selector* module selects the highest non-empty priority queue, say PQ_e and chooses the flow at the head of PQ_e, say flow i, for service. The scheduler serves the packet at the head of the queue corresponding to flow i and following the service of this packet recalculates the priority

class, z, to which flow i belongs using Equation(12). The scheduler will continue to serve the next packet from the queue of flow i until the occurrence of at least one of the following events:

(1) *A newly active flow is added to a higher priority queue :* In this case, the scheduler stops the service of flow i and begins serving the newly active flow since it belongs to a higher priority class than flow i.
(2) *Queue of flow i is empty :* In this case, flow i is removed from the head of priority queue PQ_e. Also, the $Served$ flag for flow i is set to indicate that it has received service in the current round.
(3) *The newly computed priority class of flow i, f, does not match its current priority class, e :* In this case flow i is removed from the head of priority queue PQ_e and is added to the tail of priority queue PQ_f. Note that the only exception is when $f > p$. In this case no further packets are to be scheduled from flow i during the current round since it has exhausted its allowance. Flow i is instead added to the tail of the *ActiveList*.

At any given instant of time each active flow can either be present in at most one of the p priority queues or in the *ActiveList*. Over the course of the round, as the flow consumes more and more of its allowance, it will gradually move down from the into lower priority queues until it completely exhausts its allowance following which it is added into the *ActiveList*. However during a round it is not necessary that a flow will pass through each of the p priority queues. In fact it may be possible that the only priority queue which a flow visits is the initial queue into which it is classified at the start of the round. As a result of the categorization brought about by the priority queue module in PERR, each flow uses its allowance in pieces over the course of the round. The *Organizer* reorders the sequence of transmissions to enable the flows that have not utilized a large portion of their *UnservedAllowance* to get precedence over the other flows.

The PERR scheduler also maintains two flags, *Served* and *Active* for each flow. The *Active* flag indicates whether a flow is active or not. The *Served* flag is set when the scheduler serves the first packet from a flow in the current round and remains set for the entire duration of the round indicating that the flow has been served at least once during the current round. The *Served* flags for all the flows are reset at the start of a new round. The *Served* flag prevents a flow which frequently oscillates between active and inactive periods to receive excessive service. Consider a flow which runs out of packets in the middle of a round before utilizing its entire allowance in that round. Since this flow is no longer active its *Active* flag will be reset. Assume that a new packet arrives at the flow at some time before the end of the current round. In the absence of the *Served* flag this flow would be treated as a newly active flow and its per-flow states would be reset. This would allow the flow to receive service in excess of what it would have received if it were active during the entire duration of the current round. However in PERR, the *Served* flag will be set for the flow under consideration indicating that the per-flow states for that flow

are still valid for the current round. When the flow becomes active it will be added into the appropriate priority queue using Equation (12) depending on how much of its *UnservedAllowance* was utilized when it was last active in the current round, and thus, the flow will not receive any excess service. Note that since the *Served* flags for all the flows are reset at the end of a round, accumulation of service credits from a previous round is prevented.

```
Initialize: (Invoked when the scheduler is initialized)
    MaxSC = 0;
    CurrentPriority = 0;
    Q^max = 1;
    MinNormalizedSC = MAX;
    for (i = 0; i < n; i = i + 1)
        Active_i = FALSE;
        Served_i = FALSE;

Enqueue: (Invoked when a packet arrives)
    i = QueueInWhichPacketArrives;
    if ( Active_i == FALSE) then
        if (Served_i ==FALSE) then
            SC_i = 0;
            Sent_i = 0;
        end if;
        InitializeFlow(i);
        NewPriority = ComputeNewPriority(i);
        AddToPriorityQueue(NewPriority, i);
        if (NewPriority > CurrentPriority) then
            ObtainHighestActivePriority == TRUE;
        end if
    end if;

Dequeue:
    while (TRUE) do
        if (AllPriorityQueuesEmpty == TRUE) then
            InitializeRound();
        end if;
        if(ObtainHighestActivePriority == TRUE) then
            CurrentPriority = GetHighestActivePriorityQueue;
        end if;
        i = HeadOfPriorityQueue(CurrentPriority);
        do
            TransmitPacketFromQueue(i);
            Increase Sent_i by LengthInFlitsOfTransmittedPacket;
            Served_i == TRUE;
            NewPriority == ComputeNewPriority(i);
        while ( (NewPriority == CurrentPriority) and
                (IsEmpty(Queue_i) == FALSE) and
                (ObtainHighestActivePriority == FALSE) )
        if ( (NewPriority > CurrentPriority) or
            (IsEmpty(Queue_i) == TRUE) ) then
            RemoveHeadOfPriorityQueue(CurrentPriority);
            if (IsEmpty(Queue_i) == FALSE) then
                AddToPriorityQueue(NewPriority, i);
            end if
            if (IsEmpty(CurrentPriority) == TRUE) then
                ObtainHighestActivePriority = TRUE;
            end if
        end if
    end while
```

Fig. 2. Pseudo-Code for PERR

```
InitializeRound()
    for (i = 0; i < n; i = i + 1)
        Served_i = FALSE;
    PreviousMaxSC = MaxSC;
    MaxSC = 0;
    MinNormalizedSC = MAX;
    InitializeFlow(min);
    Q^max = A_min / UA^max_min;
    ObtainHighestActivePriority == TRUE;
    while (IsEmpty(ActiveList) == FALSE) do
        flow = HeadOfActiveList;
        RemoveHeadOfActiveList;
        Sent_flow = 0;
        InitializeFlow(flow);
        NewPriority = ComputeNewPriority(flow);
        AddToPriorityQueue(NewPriority, flow);
        SC_flow = 0;
    end while;
```

Fig. 3. *InitializeRound*() Routine

```
InitializeFlow(i)
    Active_i = TRUE;
    UA_i^max = w_i(1+ PreviousMaxSC) ;
    A_i = UA_i^max − SC_i;
```

Fig. 4. *InitializeFlow*() Routine

A psuedo-code implementation of the PERR scheduling algorithm is shown in Fig. 2, consisting of the *Initialize*, *Enqueue* and *Dequeue* routines. The *Enqueue* routine is called when a new packet arrives at a flow. the *Dequeue* routine is the heart of the algorithm which schedules packets from the queues corresponding to different flows. Figs. 3–6 illustrate the pseudo-code of four routines that are used in the execution of the *Enqueue* and *Dequeue* routines. All of these routines can be easily implemented as simple hardware modules.

4 Nested Round Robin Interpretation

The primary goal of the PERR scheduler is to distribute the *UnservedAllowance* of a flow in an ERR round into several parts, so that it can be utilized in pieces over the course of the round. The Nested-DRR algorithm proposed in [17], modifies the DRR scheduler by creating a nested set of multiple rounds inside each DRR round. The Nested-DRR scheduler serves the active flows in a round robin order in these nested rounds by executing a modified version of the DRR algorithm.

The primary goal of the PERR scheduler is to distribute the *UnservedAllowance* of

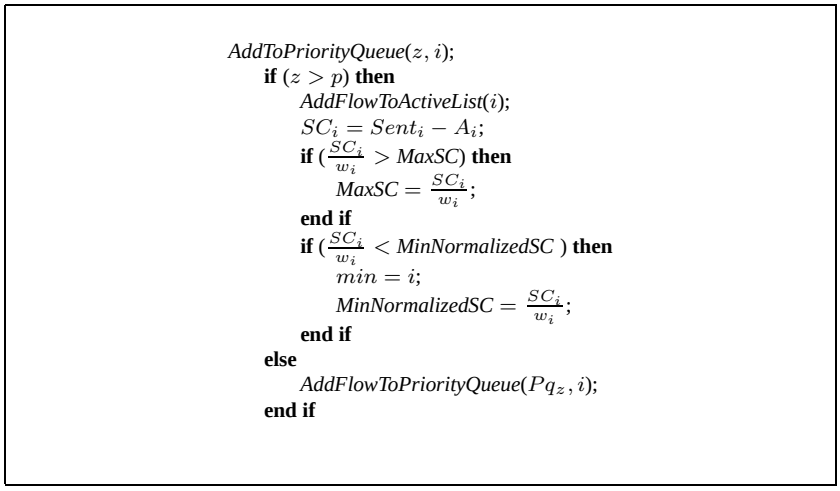

Fig. 5. *AddToPriorityQueue*() Routine

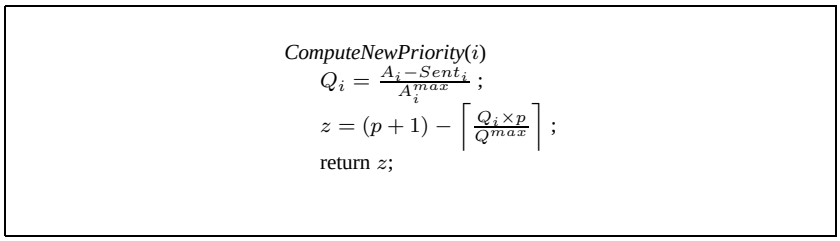

Fig. 6. *ComputeNewPriority*() Routine

a flow in an ERR round into several parts, so that it can be utilized in pieces over the course of the round. The Nested-DRR algorithm proposed in [17], modifies the DRR scheduler by creating a nested set of multiple rounds inside each DRR round. The Nested-DRR scheduler serves the active flows in a round robin order in these nested rounds by executing a modified version of the DRR algorithm.

We can hypothetically interpret the operation of the PERR scheduler as a *nested* version of ERR which is similar to Nested-DRR. This interpretation proves useful in the analysis of the latency bound of the PERR scheduler. Each round in ERR can be referred to as an *outer round*. The time interval during which the PERR scheduler serves the flows present in priority queue PQ_u during the s-th outer round is referred to as *inner round* (s, u). In effect, each outer round is split into as many inner rounds as the number of priority queues, p. Since the PERR scheduler serves the priority queues in a descending order starting at the higher priority queue PQ_p, the first inner round during outer round s will be $(s, 1)$, while (s, p) will denote the last inner round.

From Equation (11), we know that the excess service, if any, received by each flow i in the previous outer round $(s-1)$ is equal to $(1-Q_i^0(s))\,UA_i^{max}(s)$. Since $Q^{max}(s)$ represents the maximum quotient among all the flows at the start of round s, it is guaranteed that each active flow i has already utilized at least $(1 - Q^{max}(s))$-th

fraction of its ($UA_i^{max}(s)$) during its last service opportunity in the previous round $(s-1)$. The goal of the PERR scheduler is to distribute the remaining portion of each flow's maximum possible *UnservedAllowance*, $Q^{max}(s)(UA_i^{max}(s))$, equally among the p inner rounds. Let $IdealServed_i(s)$ represent the ideal service received by flow i during each inner round of the s-th outer round. $IdealServed_i(s)$ is computed as follows,

$$IdealServed_i(s) = \frac{Q^{max}(s)(UA_i^{max}(s))}{p} \tag{13}$$

Ideally, therefore, each flow i will receive exactly $IdealServed_i(s)$ amount of service in each of the p inner rounds of outer round s. In reality, however, the last packet served in an inner round from a flow may cause it to exceed its ideal service in that inner round. Just as in ERR, a *Surplus Count (SC)* is maintained for each flow which records any excess service received by the flow. The flow is penalized for this excess transmission in the subsequent inner round. When the scheduler selects a flow i for service in an inner round (s,u), its SC is incremented by $IdealServed_i(s)$. The scheduler will serve the packet at the head of flow i as long as its SC value is positive. Following the transmission of a packet, the SC corresponding to that flow is decremented by the size of the transmitted packet. Let $SC_i(s,u)$ represent the surplus count of flow i at the end of inner round (s,u). Further, let $Served_i(s,u)$ denote the actual service received by flow i in inner round (s,u). $SC_i(s,u)$ is calculated as follows,

$$SC_i(s,u) = Served_i(s,u) - (IdealServed_i(s) + SC_i(s,u-1)) \tag{14}$$

Note that, if $IdealServed_i(s)$ is less than or equal to the $SC_i(s,u-1)$, then flow i will not receive any service in inner round (s,u). Thus, a flow does not necessarily receive service in each inner round. However, the surplus count for flow i is updated at the end of each inner round using Equation (14), irrespective of whether the flow receives service in that inner round or not. In fact, it may be possible that none of the active flows receive service in an inner round. Hence, if the PERR scheduler followed a round robin service order as in Nested-DRR, then the scheduler would have a prohibitively large work complexity. However, the *Organizer* module of the PERR scheduler decides which priority queues each active flow is added into over the course of each outer round. This, in turn, determines the inner rounds in which each flow will be served. The PERR scheduler does not need to query all the active flows in a round robin order, thus leading to a low implementation complexity.

Note that the surplus count of a flow at the end of the last inner round of an outer round is the same as its surplus count at the end of the corresponding round in ERR. In other words, $SC_i(s,p)$ is the same as $SC_i(s)$. Also, note that $SC_i(s,0)$ represents the surplus count of flow i at the start of the first inner round, $(s,1)$, in outer round s. As explained earlier, we know that flow i should ideally transmit $Q^{max}(s)(UA_i^{max}(s))$ worth of data in outer round s. The remaining fraction, $(1-Q^{max}(s))$-th of the quantity $UA_i^{max}(s)$, has already been utilized in the ex-

cess service received by flow i in outer round $(s-1)$ and, therefore, is a part of $SC_i(s-1)$. To account for this already utilized portion of $UA_i^{max}(s)$, $SC_i(s,0)$ is computed as:

$$SC_i(s,0) = SC_i(s-1) - (1 - Q^{max}(s))(UA_i^{max}(s)) \tag{15}$$

It can be easily proved that Equation (5), which expresses the bounds on the surplus count, $SC_i(s)$, also holds true for $SC_i(s,u)$. Therefore, for any flow i and inner round (s,u),

$$0 \leq SC_i(s,u) \leq m-1 \tag{16}$$

Definition 5 *Let $Sent_i(s,u)$ represent the total service received by flow i since the start of the s-th outer round until the PERR scheduler has finished serving the flows in the priority queue, PQ_u.*

Note that it is not necessary that flow i was present in priority queue PQ_u during outer round s. From Equation (14), the total data served from flow i in inner round (s,u) is,

$$Served_i(s,u) = IdealServed_i(s) + SC_i(s,u) - SC_i(s,u-1) \tag{17}$$

$Sent_i(s,u)$ is calculated as follows:

$$Sent_i(s,u) = \sum_{w=1}^{w=u} Served_i(s,w) \tag{18}$$

Substituting for $Served_i(s,w)$ from Equation (17) in Equation (18), we have,

$$Sent_i(s,u) = u(IdealServed_i(s)) + SC_i(s,u) - SC_i(s,0) \tag{19}$$

$Sent_i(s,u)$ will be positive only if $u(IdealServed_i(s))$ is greater than $SC_i(s,0)$. Otherwise it indicates that flow i has not received any service until the end of the (s,u)-th inner round. However flow i is guaranteed to receive service in at least one inner round during the s-th outer round. Using Equations (13), (15) and (19) we have,

$$Sent_i(s,u) = \left(\frac{u}{p}\right) Q^{max}(s)\, UA_i^{max}(s) + SC_i(s,u) - SC_i(s-1) \tag{20}$$

Definition 6 *Define $Sent_i(s)$ as the total service received by flow i in outer round s.*

Note that $Sent_i(s,p)$ represents the service received by flow i when the scheduler has finished serving the flows in priority queue PQ_p which in fact equals $Sent_i(s)$. Substituting $u = p$ in Equation (20) and using Equation(8), we get,

$$Sent_i(s) = w_i(1 + MaxSC(s-1)) + SC_i(s) - SC_i(s-1) \tag{21}$$

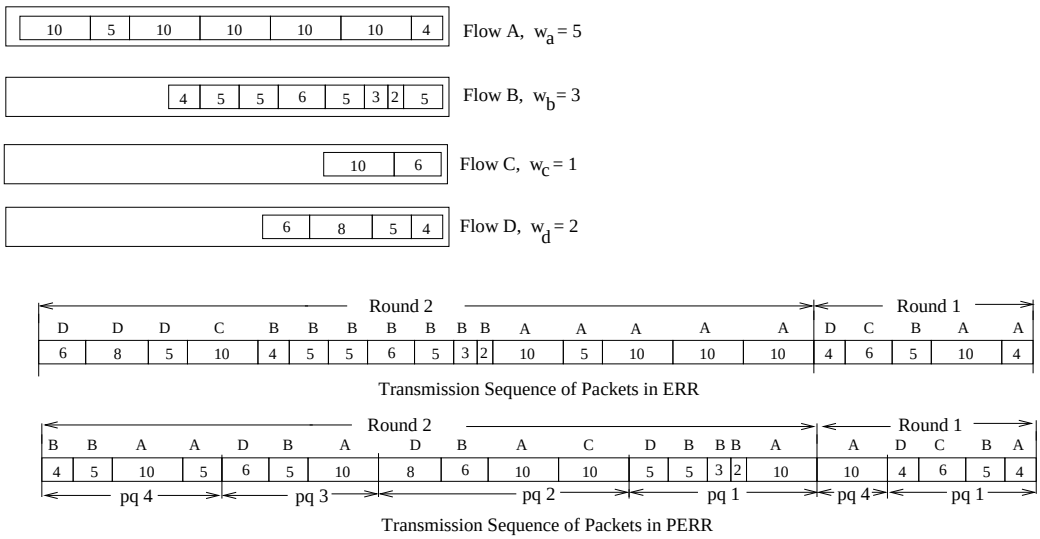

Fig. 7. Comparison of the transmission sequence of packets in ERR and PERR

Hence the total service received by flow i in an outer round in PERR is identical to the service received by flow i in the corresponding round in ERR.

Ideally, during the normal operation of the PERR scheduler, the p inner rounds in outer round s follow a strictly sequential order starting at inner round $(s, 1)$ and ending at round (s, p). However, in certain situations, it is possible to interrupt the sequential ordering. Let us assume that a flow j becomes active for the first time in outer round s while the PERR scheduler is serving a flow k at the head of priority queue PQ_d. Since the quotient for flow j is equal to 1, it will be added into priority queue PQ_1 which has the highest priority among all the priority queues. Upon finishing the transmission of the current packet from flow k, the PERR scheduler temporarily suspends the service of flow k and starts serving flow j which is at the head of queue PQ_p. The PERR scheduler will keep serving flow j until it is added either into priority queue PQ_d or some other queue with lower priority than PQ_d. The scheduler will then resume service of the k-th flow. Note that the service received by flow i while it is present in queue PQ_p is part of the inner round (s, p) even though it is not contiguous with the time interval during which the PERR scheduler served the flows present in queue PQ_p at the start of the outer round s. Also, the inner round (s, d) will not extend over a continuous time interval because it will be interposed by the entire service received by flow i since the time it became active until its addition into priority queue PQ_d or a lower priority queue. It is, therefore, not necessary that the inner rounds in an outer round should sequentially follow one another and that the flows which receive service in an inner round should be served in succession. However, note that this disruption of the otherwise sequential service can only be caused due to a new flow becoming active during the execution of that outer round.

Fig. 7 compares the transmission sequence in the first two rounds of execution of the ERR and PERR schedulers for the given input pattern and flow weights. In

the PERR scheduler, the flows are classified into 4 classes corresponding to the 4 priority queues. At the start of inner round $(1, 1)$ all the flows are present in priority queue PQ_1. After receiving service in this inner round, flow A is added into PQ_4. However the other 3 flows exceed their allowance in this inner round and are therefore added into the *ActiveList*. Note that, in the second outer round, unlike the ERR scheduler where the flows B, C and D have to wait for their turn in the round robin order to receive service, flows B and D start receiving service in the inner round $(2, 1)$ whereas flow C is served for the first time in inner round $(2, 2)$.

5 Analytical Results

In this section, we present analytical results on the fairness, latency properties and the work complexity of PERR.

5.1 Latency Bound of PERR

In deriving an upper bound on the latency of PERR we use the concept of Latency-Rate ($\mathcal{LR}$) servers first proposed in [5]. We now define the notion of a busy period, essential for developing the concept of the $\mathcal{LR}$ servers.

Definition 7 *A busy period of a flow is defined as the maximal time interval during which the flow is active if it is served at exactly its reserved rate.*

Note that the active period of a flow is different from the busy period in the sense that it reflects the actual behavior of the scheduler where the instantaneous service offered to the flow varies according to the number of active flows.

Definition 8 *Let $Sent_i(t_1, t_2)$ represent the amount of service received by flow i during the time interval (t_1, t_2).*

Note that this notation is identical to the one used in Definition 5, except for the type of the parameters. Thus, $Sent_i(\beta, \gamma)$ is to be interpreted according to whether β and γ represent two instants of time or (β, γ) denotes an inner round in the execution of the PERR scheduler. In the following, the context will always make clear as to how $Sent_i(\beta, \gamma)$ should be interpreted.

Let the time instant α_i be the start of a busy period for flow i. Let $t > \alpha_i$ be such that flow i is continuously busy during the time interval (α_i, t). Let $S_i(\alpha_i, t)$ be the number of bits belonging to packets in flow i that arrive after time α_i and are scheduled during the time interval (α_i, t). Note that, during this time interval the scheduler may still be serving packets from a previous busy period, and hence $S_i(\alpha_i, t)$ is not necessarily the same as $Sent_i(\alpha_i, t)$.

Definition 9 *The latency of a flow is defined as the minimum non-negative constant Θ_i that satisfies the following for all possible busy periods of the flow,*

$$S_i(\alpha_i, t) \geq max\{0, \rho_i(t - \alpha_i - \Theta_i)\} \tag{22}$$

As defined in [5], a scheduler which satisfies Equation (22) for some non-negative constant value of Θ_i is said to belong to the class of Latency Rate ($\mathcal{LR}$) servers. In practice, however it is easier to analyze scheduling algorithms based on the active period of a flow. Let τ_i be an instant of time when flow i becomes active. Let $t > \tau_i$ be some time instant such that the flow is continuously active during the time interval (τ_i, t) Let Θ_i' be the smallest non-negative number such that the following equation is satisfied for all t.

$$Sent_i(\alpha_i, t) \geq \max\{0, \rho_i(t - \alpha_i - \Theta_i')\} \tag{23}$$

Even though (τ_i, t) may not be a continuously busy period for flow i, it is proved in [5], that the latency as defined by (22) is bounded by Θ_i'.

Theorem 1 *The PERR scheduler belongs to the class of $\mathcal{LR}$ servers, with an upper bound on the latency Θ_i for flow i given by,*

$$\Theta_i \leq \frac{\frac{(W-w_i)m}{p} + (n-1)(m-1)}{r} \tag{24}$$

where n is the total number of active flows, p is the number of priority queues, r is the transmission rate of the output link and W is the sum of the weights of all the flows.

Proof. Since the latency of a $\mathcal{LR}$ server can be estimated based on its behavior in the flow active periods, we will prove the theorem by showing that,

$$\Theta_i' \leq \frac{\frac{(W-w_i)m}{p} + (n-1)(m-1)}{r}.$$

Let τ_i be the time instant when flow i becomes active. To prove the statement of the theorem we consider a time interval (τ_i, t), where $t > \tau_i$, during which flow i is continuously active. We first obtain the lower bound on the total service received by flow i during the time interval under consideration. Then we express the lower bound in the form of Equation (23) to derive the latency bound.

In [19] it has been proved that to obtain a tight upper bound on the latency of the ERR scheduler, we must consider an active period (τ_i, t) such that τ_i coincides with the beginning of the service opportunity of a flow and t belongs to the set of time instants at which the scheduler begins serving flow i. We can easily prove that the same conditions apply for proving the upper bound on the latency of the PERR scheduler. Let $\tau_i^{(e,f)}$ be the time instant marking the start of the service of flow i

when flow i is at the head of priority queue PQ_f in round e. In other words, this time instant represents the start of the service opportunity of flow i in inner round (e, f). Therefore, in trying to determine the latency bound of the PERR, we need to only consider time interval $(\tau_i, \tau_i^{(e,f)})$ for all (e, f).

The first step in proving the latency bound involves determining the upper bound on the size of the time interval under consideration. Note that the time instant τ_i may or may not coincide with the start of a new round. Let k_0 be the round which is in progress at time instant τ_i or which starts exactly at time instant τ_i. Let t_h mark the start of the round $(k_0 + h)$. In either case, flow i will be able to transmit at least $A_i(k_0)$ worth of data over the course of the k_0-th round. If flow i becomes active when the round k_0 is in progress, i.e. when $\tau_i < t_0$ then the service received during the interval (t_0, τ_i) will be excluded from the time interval under consideration. The time interval $(\tau_i, \tau_i^{(e,f)})$ will be maximal only if the time instant τ_i coincides with t_0, the start of the k_0-th round. Hence we assume that τ_i coincides with the start of the the k_0-th round. Fig. 8 illustrates the interval under consideration assuming that (e, f) is equal to $(k_0 + k, v)$. Note that in Fig. 8, $OR(e)$ represents the e-th outer round in the execution of the PERR scheduler and $IR(e, f)$ denotes the inner round (e, f).

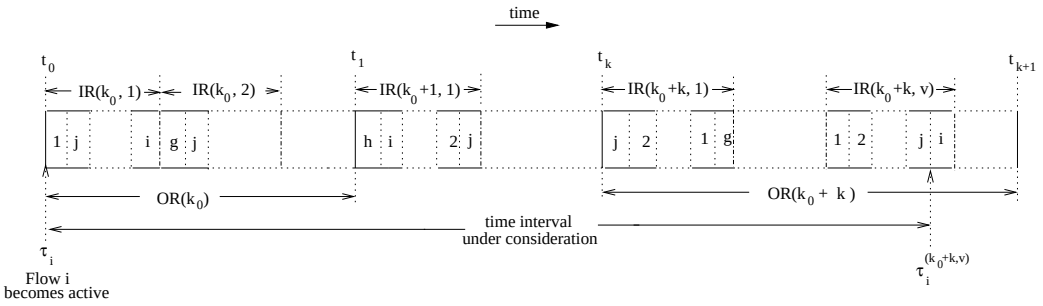

Fig. 8. An illustration of the time interval under consideration

The time interval under consideration, $(\tau_i, \tau_i^{(k_0+k,v)})$, can be split into two sub-intervals:

(1) (τ_i, t_k): This sub-interval includes k rounds of execution of the PERR scheduler starting at round k_0. Consider the time interval (t_h, t_{h+1}) when round $(k_0 + h)$ is in progress. Summing Equation (21) over all n flows,

$$t_{h+1} - t_h = \frac{W}{r}(1 + MaxSC(k_0 + h - 1)) + \frac{1}{r}\sum_{j=1}^{n}\{SC_j(k_0 + h) - SC_j(k_0 + h - 1)\}$$

Summing the above over k rounds beginning with round k_0,

$$t_k - \tau_i = \frac{W}{r}(k) + \frac{W}{r}\sum_{h=0}^{k-1} MaxSC(k_0+h-1) + \frac{1}{r}\sum_{j=1}^{n}\{SC_j(k_0+k-1) - SC_j(k_0-1)\} \quad (25)$$

(2) $(t_k, \tau_i^{(k_0+k,v)})$: This sub-interval includes the part of the (k_0+k)-th round prior to the start of the service of flow i when it is at the head of priority queue PQ_v. In the worst-case flow i will be the the last flow to receive service among all other flows which are present in priority queue PQ_v. In this case, during the sub-interval under consideration, the service received by flow i equals $Sent_i(k_0+k, v-1)$ whereas the service received by each flow j among the other $n-1$ flows equals $Sent_i(k_0+k, v)$. Note that if v equals 1 then flow i does not receive service in this sub-interval. Hence summing $Sent_i(k_0+k, v-1)$ and $Sent_j(k_0+k, v)$ for each flow j such that $1 \leq j \leq n, j \neq i$ and using Equation (20), we have,

$$\begin{aligned}\tau_i^{(k_0+k,v)} - t_k = {} & \frac{1}{r}\sum_{\substack{j=1\\ j\neq i}}^{n}\left(\frac{v}{p}\right) Q^{max}(k_0+k)\, UA_j(k_0+k) \\ & + \frac{1}{r}\left(\frac{v-1}{p}\right) Q^{max}(k_0+k)\, UA_i(k_0+k) \\ & + \frac{1}{r}\sum_{\substack{j=1\\ j\neq i}}^{n}(SC_j(k_0+k, v) - SC_j(k_0+k-1)) \\ & + \frac{1}{r}(SC_i(k_0+k, v-1) - SC_i(k_0+k-1)) \end{aligned} \quad (26)$$

To simplify the analysis we introduce a new variable Ω, such that,

$$\Omega = Q^{max}(k_0+k)(1 + MaxSC(k_0+k-1)) \quad (27)$$

Using Equation (6) and Corollary 1 in Equation (27), we get,

$$0 < \Omega \leq m \quad (28)$$

Combining Equations (25) and (26) and using Equations (5), (8) and (27), we have,

$$\begin{aligned}\tau_i^{(k_0+k,v)} - \tau_i \leq {} & \frac{W}{r}k + \frac{W}{r}\sum_{h=0}^{k-1} MaxSC(k_0+h-1) + \frac{1}{r}\sum_{\substack{j=1\\ j\neq i}}^{n}\left(\frac{v}{p}\right) w_j\Omega \\ & + \frac{1}{r}\left(\frac{v-1}{p}\right) w_i\Omega + \frac{1}{r}(n-1)(m-1) + \frac{1}{r}SC_i(k_0+k, v-1) \end{aligned} \quad (29)$$

Solving for k and using the fact that W is the sum of the weights of all the n flows,

$$k \geq (\tau_i^{(k_0+k,v)} - \tau_i)\frac{r}{W} - \sum_{h=0}^{k-1} MaxSC(k_0+h-1) - \frac{1}{W}\left(\frac{v}{p}\right)(W-w_i)\Omega - \frac{1}{W}\left(\frac{v-1}{p}\right)w_i\Omega - \frac{1}{W}(n-1)(m-1) - \frac{1}{W}SC_i(k_0+k,v-1) \quad (30)$$

Note that the total data transmitted by flow i during the time interval under consideration can be expressed as the following summation,

$$Sent_i(\tau_i, \tau_i^{(k_0+k,v)}) = Sent_i(\tau_i, t_k) + Sent_i(t_k, \tau_i^{(k_0+k,v)}) \quad (31)$$

As explained, earlier $Sent_i(t_k, \tau_i^{(k_0+k,v)})$ is the same as $Sent_i(k, v-1)$. $Sent_i(\tau_i, t_k)$ can be obtained by summing Equation (21) over k rounds starting at round k_0. Substituting the result and Equation (20) in Equation (31) and using Equation (27) and the fact that the surplus count of a newly active flow is equal to 0, we have,

$$Sent_i(\tau_i, \tau_i^{(k_0+k,v)}) = w_i k + w_i \sum_{h=0}^{k-1} MaxSC(k_0+h-1) + \left(\frac{v-1}{p}\right)w_i\Omega + SC_i(k_0+k,v-1) \quad (32)$$

Using (30) to substitute for k in (32), we get,

$$Sent_i(\tau_i, \tau_i^{(k_0+k,v)}) \geq \frac{w_i r}{W}(\tau_i^{(k_0+k,v)} - \tau_i) + \left(\frac{v-1}{p}\right)w_i\Omega\left(\frac{W-w_i}{W}\right) - \frac{w_i}{W}\left(\frac{v}{p}\right)(W-w_i)\Omega - \frac{w_i}{W}(n-1)(m-1) - SC_i(k_0+k,v-1)\left(\frac{w_i}{W}-1\right)$$

Simplifying further we get,

$$Sent_i(\tau_i, \tau_i^{(k_0+k,v)}) \geq \frac{w_i r}{W}\left((\tau_i^{(k_0+k,v)} - \tau_i) - \frac{1}{r}\left(\frac{W-w_i}{p}\right)\Omega - \frac{1}{r}(n-1)(m-1)\right) - SC_i(k_0+k,v-1)\left(\frac{w_i}{W}-1\right) \quad (33)$$

Now, since the reserved rates are proportional to the weights assigned to the flows as given by (1), and since the sum of the reserved rates is no more than the link rate r, we have,

$$\rho_i \leq \frac{w_i}{W}r \quad (34)$$

Using Equation (29) it can be verified that the multiplicand of $\frac{w_i r}{W}$ in Equation (33) is always positive. Substituting for $\frac{w_i r}{W}$ from Equation (34) in Equation (33) we

have,

$$Sent_i(\tau_i, \tau_i^{(k_0+k,v)}) \geq \rho_i \Bigg((\tau_i^{(k_0+k,v)} - \tau_i) - \frac{1}{r} \left(\frac{W - w_i}{p} \right) \Omega - \frac{1}{r}(n-1)(m-1) \Bigg) - SC_i(k_0+k, v-1) \left(\frac{w_i}{W} - 1 \right) \quad (35)$$

Noting that the latency reaches the upper bound when $SC_i(k_0 + k, v - 1)$ equals 0 and Ω equals m, we get,

$$Sent_i(\tau_i, \tau_i^{(k_0+k,v)}) \geq max \Bigg\{ 0, \rho_i \Bigg(\tau_i^{(k_0+k,v)} - \tau_i - \left(\frac{(\frac{W-w_i}{p})m + (n-1)(m-1)}{r} \right) \Bigg) \Bigg\} \quad (36)$$

As discussed earlier, flow i will experience its worst latency during an interval $(\tau_i, \tau_i^{(e,f)})$ for some inner round (e, f). Therefore, from Equation (36), the statement of the theorem is proved. ■

We now proceed to show that the latency bound given by Theorem 1 is tight by illustrating a case where the bound is actually met. Assume that a flow i becomes active at time instant τ_i, which also coincides with the start of a certain round k_0. Assume that for any time instant t, $t \geq \tau_i$, a total of n flows, including flow i, are active. Also, assume that the summation of the reserved rates of all the n flows equals the output link transmission rate, r. Hence, $\rho_i = \frac{w_i}{W} r$. Since flow i became active at time τ_i, its surplus count at the start of round k_0 is 0. Let the surplus count of all the other flows at the start of round k_0 be equal to 0. Assume that, a flow l which is not active after time τ_i and hence is not included in the n flows, was active during the k_0-th round. Assume that flow l exceeded its allowance by $(m-1)$ in its last service opportunity in round $(k_0 - 1)$, leading to a value of $MaxSC(k_0 - 1)$ equal to $(m-1)$. Since the surplus counts of all the n active flows are equal to 0, the *Unserved Allowance Quotient* for all the flows at the start of the k_0-th round will be equal to unity. Hence $Q^{max}(k_0)$ will be equal to 1 and all the n flows will be added into the priority queue PQ_p at the start of the round. Assume that flow i is the last flow to be added into this queue. From Equations (16), (6) and (17), any given flow j can transmit a maximum of $w_j(\frac{m}{p}) + (m-1)$ bits during its service opportunity in an inner round. In the worst case, before flow i is served by the PERR scheduler, each of the other $(n-1)$ flows will receive this maximum

service. Hence, the cumulative delay until flow i receives service is given by,

$$D = \frac{(\sum_{j \neq i} w_j)(\frac{m}{p}) + (n-1)(m-1)}{r}$$
$$= \frac{(\frac{W - w_i}{p})m + (n-1)(m-1)}{r}$$

Noting that $S_i(\tau_i, \tau_i + D)$ equals zero, it is readily verified that the bound is exactly met at time $t = \tau_i + D$.

5.2 Fairness Bound of PERR

In our fairness analysis, we use the popular metric, *Relative Fairness Bound (RFB)* first proposed in [8]. The RFB is defined as the maximum difference in the normalized service received by any two flows over all possible intervals of time.

Definition 10 *Given an interval (t_1, t_2), the* Relative Fairness, $RF(t_1, t_2)$ *for this interval is defined as the maximum value of $|\frac{Sent_i(t_1,t_2)}{w_i} - \frac{Sent_j(t_1,t_2)}{w_j}|$ over all pairs of flows i and j that are active during this time interval. Define the* Relative Fairness Bound (RFB) *as the maximum of $RF(t_1, t_2)$ over all possible time intervals (t_1, t_2).*

The *Absolute Fairness Bound (AFB)* is a related measure of fairness which captures the upper bound on the difference between the normalized service under the current scheduler and that it would receive with the ideal GPS scheduler [1]. It has been shown in [20], that the AFB and RFB are related to each other by a simple relationship. Hence, we only discuss the RFB in this report.

Theorem 2 *For any execution of the PERR scheduling discipline, $RFB < 2m + \frac{2m}{p}$*

Proof. In [16], while analyzing the fairness properties of ERR, we have proved that a tight upper bound on the the RFB of ERR can be obtained by considering only a subset of all possible time intervals. This subset is the set of all time intervals bounded by time instants that coincide with the start of the service opportunities of flows. It can be easily verified that to prove the RFB of the PERR scheduler we need to consider a time interval (t_1, t_2) such that both the time instants t_1 and t_2 coincide with the start of a service opportunity of a flow in an inner round.

Consider any two flows i and j that are active in the time interval between the time instants t_1 and t_2. Let (k_0, f) and $(k_0 + k, g)$ be the inner rounds which are in progress at time instants t_1 and t_2 respectively. Let time instant $t_{(h,v)}$ mark the start of inner round $(k_0 + h, v)$. In other words, $t_{(k_0,f)} < t_1 < t_{(k_0,f+1)}$ and $t_{(k_0,g)} < t_2 < t_{(k_0,g+1)}$. It may be possible that, if flow j receives service in the inner round (k_0, f), then it does so in the time interval $(t_{(k_0,f)}, t_1)$. Unlike the ERR scheduler, in PERR

if flow j is served before flow i in a certain inner round then, it is not necessary that the same order of service is followed in each of the following inner rounds. Hence on a similar note, it may be possible that if flow j receives service in the inner round $(k_0 + k, g)$, then that service is also not included in the time interval under consideration. Hence $Sent_i(t_1, t_2)$ and $Sent_j(t_1, t_2)$ can be evaluated as follows:

$$\begin{aligned} Sent_i(t_1, t_2) &= Sent_i(k_0) - Sent_i(k_0, f-1) \\ &\quad + \sum_{h=1}^{h=k-1} Sent_i(k_0 + h) + Sent_i(k_0 + k, g) \\ Sent_j(t_1, t_2) &= Sent_j(k_0) - Sent_j(k_0, f) \\ &\quad + \sum_{h=1}^{h=k-1} Sent_j(k_0 + h) + Sent_j(k_0 + k, g-1) \end{aligned}$$

Using Equations (8), (20) and (21) in the above and simplifying, we get,

$$\begin{aligned} Sent_i(t_1, t_2) = &\left(\frac{f}{p} Q^{max}(k_0)\right) UA_i^{max}(k_0) + k \sum_{h=1}^{h=k-1} UA_i^{max}(k_0 + h) \\ &+ \left(\frac{g}{p} Q^{max}(k_0 + k)\right) UA_i^{max}(k_0 + k) \\ &+ SC_i(k_0 + k, g) - SC_i(k_0, f-1) \end{aligned} \tag{37}$$

$$\begin{aligned} Sent_j(t_1, t_2) = &\left(\frac{f-1}{p} Q^{max}(k_0)\right) UA_j^{max}(k_0) + k \sum_{h=1}^{h=k-1} UA_j^{max}(k_0 + h) \\ &+ \left(\frac{g-1}{p} Q^{max}(k_0 + k)\right) UA_j^{max}(k_0 + k) \\ &+ SC_j(k_0 + k, g-1) - SC_j(k_0, f) \end{aligned} \tag{38}$$

Without loss of generality, we can assume that in the interval (t_1, t_2) flow i receives more service as compared to flow j. The normalized service for each of the flows can be obtained by dividing the above two equations by their respective weights. Subtracting the normalized service of flow j from that of flow i using Equations (37) and (38) we have,

$$\begin{aligned} \frac{Sent_i(t_1, t_2)}{w_i} - \frac{Sent_j(t_1, t_2)}{w_j} &= \frac{UA_i^{max}(k_0) Q^{max}(k_0)}{w_i\, p} \\ &\quad + \frac{UA_j^{max}(k_0 + k) Q^{max}(k_0 + k)}{w_j\, p} + \frac{SC_i(k_0 + k, g)}{w_i} \\ &\quad - \frac{SC_i(k_0, f-1)}{w_i} + \frac{SC_j(k_0 + k, g-1)}{w_j} - \frac{SC_j(k_0, f)}{w_j} \end{aligned}$$

Simplifying the above using Equations (6), (8), (16) and Corollary 1, the statement of the theorem is proved. ■

Consider an execution of the PERR scheduler over n flows. The work involved in processing each packet at the scheduler involves two parts: enqueueing and dequeueing. Hence the work complexity of a scheduler is defined as the order of time complexity, with respect to n of enqueueing and then dequeueing a packet for transmission [12, 16]. Note that n, the number of flows competing for a link can be of the order of tens of thousands of flows in backbone routers. Hence it is desirable that the work complexity should be as independent as possible of n.

Theorem 3 *The worst-case work complexity of the PERR scheduler is* $O(\log p)$.

Proof. The time complexity of enqueueing a packet is the same as the time complexity of the *Enqueue* routine in Fig. 2, which is executed whenever a new packet arrives at a flow. Identifying the flow at which the packet arrives is an $O(1)$ operation. If the *Active* flag is not set for the flow i, the *Ideal Allowance Utilization* for the flow is calculated which in turn determines the priority queue into which the flow should be added. Also, if this priority queue is of a higher priority than the priority queue which the PERR scheduler is serving, a flag is set to indicate that after completing the transmission of the current packet, the scheduler should start serving packets from the newly active flow. The addition of an item to the tail of a linked-list data structure and conditionally setting a flag are both $O(1)$ operations.

Let us now consider the time complexity of dequeueing a packet. Assume that the PERR scheduler is serving flows from the priority queue PQ_g. Note that at least one packet is transmitted from each of the flows that are present in PQ_g. The operations involved in serving flows in this priority queue include determining the next flow to be served, removing this flow from the head of the priority queue and possibly adding it into some other priority queue or the ActiveList. All these operations can be executed in $O(1)$ time. Additionally the PERR scheduler may need to update certain per-flow variables which can be easily done in constant time. However once the queue PQ_g is empty the PERR scheduler needs to determine the highest non-empty priority queue. To aid in this, the PERR scheduler maintains a linked list of the identifiers of all the non-empty priority queues. This linked list is sorted in descending order of priority with the head of the list pointing to the highest non-empty priority queue. The complexity of inserting a new identifier into this sorted linked list is $O(\log p)$ where p represents the total number of priority queues. To select the highest non-empty priority queue, the PERR scheduler simply has to read the identifier at the head of this sorted list which can be done in $O(1)$ time. Hence the overall time complexity of this operation is $O(\log p)$ A similar situation arises when a flow is added into a priority queue which has a higher priority than the current priority queue being served by the PERR scheduler. However if all the priority queues are empty it is an indication that the current round has come to an end. In this case, prior to the start of the subsequent round, the *Organizer* module

has to classify all the flows present in the *ActiveList* into the p priority queues which requires $O(n)$ time. However since each of the n flows is guaranteed to transmit at least one packet, the overall complexity of this operation is $O(1)$.

Note that the PERR scheduler needs to sort the non-empty priority queues only in the two special cases discussed above, unlike the sorted-priority algorithms like WFQ and SCFQ where these sorting operations need to be executed prior to each packet transmission resulting in a $O(\log n)$ work complexity where n is the number of active flows. Also, since $n \gg p$, the work complexity of the PERR scheduler is always lower than that of the sorted-priority schedulers. Hence the worst-case work complexity of the PERR scheduler is $O(\log p)$ resulting in an efficient hardware implementation. ■

6 Comparison with Other Schedulers

Table 1 summarizes the work complexity, fairness and latency bounds of several guaranteed-rate scheduling disciplines that belong to the class of ($\mathcal{LR}$) servers. In this table, n represents the number of active flows and p represents the number of priority queues in Pre-order DRR and PERR. The peak rate of the output link is denoted by r. M is the size of the largest packet that may potentially arrive during the execution of a scheduling algorithm. Recall that m is the size of the largest packet that *actually* arrives during the execution of the scheduler. Usually, in most networks including the Internet, $M \gg m$ since the majority of packets are much smaller than the largest possible packet [21, 22]. The properties of all the frame-based scheduling disciplines are derived in [23]. The latency bounds of DRR and Pre-order DRR have been analyzed in [24] and [25]. The RFB and latency bound of ERR have been analyzed in [16, 19]. The expression for the latency bound of Nested-DRR, as proved in [17], is extremely complex and hence does not allow for an easy comparison with the other schedulers under consideration. Hence, in order to gain a quick understanding of the differences in the latency bounds of Nested-DRR and PERR, in our comparison we include the latency bound of Nested-DRR at two boundary conditions. In the first case, we consider the latency bound of a flow whose reserved rate is much lower than that of the other flows sharing the same output link ($\rho_i \ll \rho_j, \forall j \in n, j \neq i$). In the second case, we consider the opposite end of the spectrum, i.e., the latency bound of a flow whose reserved rate is much greater than the other flows multiplexed on the same link ($\rho_i \gg \rho_j, \forall j \in n, j \neq i$). In [17], an expression for the latency of a low-rate flow has been derived and it has also been shown that the latency of a high-rate flow is marginally lower than that of the DRR scheduler. For simplicity we assume the latter to be equal to DRR.

Table 1: A Comparison between Scheduling Disciplines

Scheduler	Complexity	Fairness	Latency Bound for flow i
GPS [6]	—	0	0
Weighted Fair Queueing [7]	$O(\log n)$	$O(n)$	$\frac{m}{r} + \frac{m}{\rho_i}$
Self Clocked Fair Queueing [8]	$O(\log n)$	$2m$	$\frac{(n-1)m}{r} + \frac{m}{\rho_i}$
Virtual Clock [26]	$O(\log n)$	∞	$\frac{m}{r} + \frac{m}{\rho_i}$
Frame-based Fair Queueing [10]	$O(\log n)$	$2M + m$	$\frac{m}{r} + \frac{m}{\rho_i}$
Deficit Round Robin [12]	$O(1)$	$M + 2m$	$\frac{1}{r}\left\{(W - w_i)Q_{min} + (m-1)\left(\frac{W}{w_i} + n - 2\right)\right\}$
Elastic Round Robin [16, 19]	$O(1)$	$3m$	$\frac{1}{r}\{(W - w_i)Q_{min} + (m-1)(n-1)\}$
Nested-DRR [17]	$O(1)$	$M + 2m$	*Low-rate:* $\frac{1}{r}\left\{(n-1)Q_{min} + (m-1)\left(\frac{W}{w_i} + n - 2\right)\right\}$ *High-rate:* $\frac{1}{r}\left\{(W - w_i)Q_{min} + (m-1)\left(\frac{W}{w_i} + n - 2\right)\right\}$
Pre-order DRR [18]	$O(\log p)$	$\frac{2M}{p} + 2m$	$\frac{1}{r}\left\{\frac{(W - w_i)Q_{min}}{p} + (m-1)\left(\frac{W}{w_i} + n - 2\right)\right\}$
Prioritized ERR	$O(\log p)$	$\frac{2m}{p} + 2m$	$\frac{1}{r}\left\{\frac{(W - w_i)Q_{min}}{p} + (m-1)(n-1)\right\}$

The latency of the GPS [6] scheduler is zero since a newly active flow begins receiving service instantaneously at its reserved rate. However, GPS, while ideally fair, is unimplementable. The sorted priority schedulers such as Weighted Fair Queueing (WFQ) [7], Frame-based Fair Queueing (FFQ) [10] and Virtual Clock [26] have a low value of the latency bound. The work complexity of all these schedulers, however, is a function of the total number of flows. In fact, only ERR, DRR and Nested-DRR have a work complexity of $O(1)$. ERR has better fairness properties and a lower latency bound than DRR. As discussed earlier, ERR, DRR and Nested-DRR are round-robin schedulers and hence suffer from the characteristic limitations of all round robin schedulers. Both Pre-order DRR and PERR overcome these drawbacks in the DRR and ERR schedulers respectively and also have a low work complexity of $O(\log p)$ which is independent of the number of flows. As explained earlier, the basic principle of the PERR algorithm is similar to the Pre-order DRR algorithm [18]. The Pre-order DRR scheduler alters the transmission sequence of the packets in each DRR round based on the quantum utilization of each flow. The PERR scheduler similarly changes the sequence in which packets are transmitted in an ERR round depending on the utilization of the maximum possible allowance

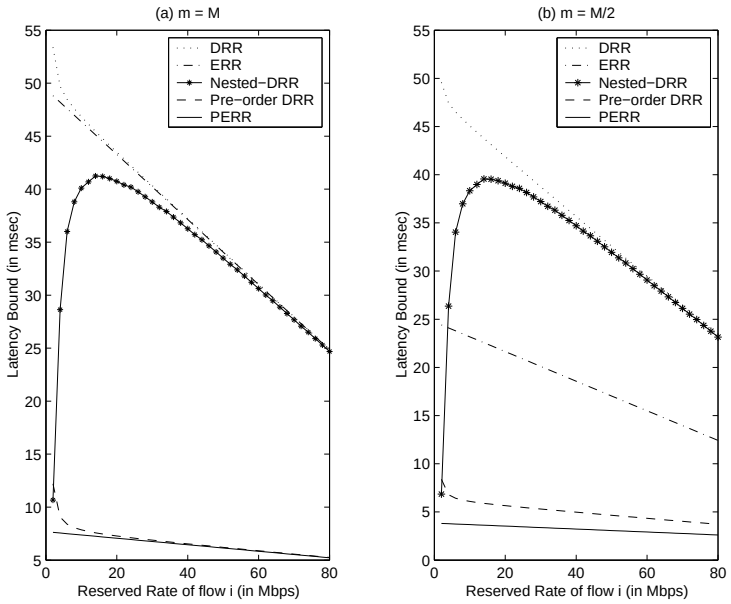

Fig. 9. Comparison of the latency bounds

of each flow in that round.

There is however an important difference between these two schedulers. At the start of a round, the Pre-order DRR scheduler has to classify all the packets that will be transmitted by the active flows in that round into the priority queues prior to starting the transmission of the packets. On the contrary, the PERR scheduler simply has to classify the flows present in the *ActiveList* into its priority queues before the start of the round. It has been shown in [19] that ERR has a couple of important advantages over DRR. Now since the PERR and Pre-order DRR algorithms are modifications of ERR and DRR, respectively, PERR inherits those advantages over Pre-order DRR. The following lists these advantages:

- *Lower Buffer Requirement :* Since the priority queues in PERR simply need to contain the flow identifiers they can be much smaller in size as compared to those in Pre-order DRR which have to buffer all the packets that will be transmitted in the current round.
- *Reduced Round Start Delay :* Classifying all the packets that are to be transmitted in the round into the priority queues requires considerably more time than simply sorting the flow identifiers. Thus, the delay incurred by the PERR scheduler at the start of the round is much less in comparison to that incurred by the Pre-order DRR scheduler.
- *Improved Latency and Fairness Characteristics:* Table 1 illustrates that PERR has better fairness properties and a lower latency bound than Pre-order DRR, especially considering that M is typically much greater than m.
- *Adaptability in other contexts:* Unlike DRR and Pre-order DRR, the PERR scheduler does not require the knowledge of the transmission time of each packet. As a result the scheduler can be used in other networks such as wormhole networks,

where the transmission time of a packet depends not only on the size of the packet but also the downstream congestion. Similarly, PERR may be used in ATM networks transmitting IP packets over AAL5, where the end of the packet is not known until the last ATM cell of the packet arrives.

Note that the latency bound of a $\mathcal{LR}$ server is an extremely important QoS parameter since it has a direct influence of the the size of the playback buffers needed at the receiver for real-time communication applications. In order that the reader can fully appreciate the improvement in the latency characteristics of PERR, we compare the latency bound of PERR with other efficient scheduling disciplines of equivalent work complexity such as DRR, ERR, Nested-DRR and Pre-order DRR within the context of an example. Note that for this comparison we use the actual latency bound for Nested-DRR as proved in [17].

Let us assume that a total of 100 flows are multiplexed on an output link with a transmission rate, r of 150 Mbps. Assume that M is equal to 576 bytes, equal to the minimum value of the Maximum Transmission Unit (MTU) required of all networks. Assume that ρ_{min} is equal to 0.1Mbps and that the output link is completely utilized, i.e. $\sum_{i=1}^{n} \rho_i = r$. Note that this implies that the sum of all the weights, W, is equal to $150/0.1 = 1500$. Let the number of priority queues in the priority queue module of Pre-order DRR and PERR be equal to 10, i.e., $p = 10$. We compare the latency bounds of the afore-mentioned schedulers for flow i as a function of its reserved rate, ρ_i, for two values of m : (a) $m = M$, (b) $m = M/2$. Fig. 9 illustrates a plot of these latency bounds of flow i for both the values of m. Note that latency bounds of all the schedulers under consideration depend on the sum of the weights of all the flows but not on the distribution of the weights among all the flows other than flow i. Therefore, the weights of the flows other than flow i are not discussed in the context of this illustration. It can be easily seen that PERR has the lowest latency bound among all the schedulers under consideration. The improvement in the latency of PERR is even more apparent when $m < M$.

7 Simulation Results

In this section, we first present a brief discussion on a recently proposed new measure of fairness which captures the *instantaneous* behavior of a scheduler. A more detailed presentation of this measure may be found in [27]. We then present some simulation results comparing the performance of PERR based on this new metric with other efficient schedulers.

Table 1 lists the RFBs of the schedulers. However, measures such as the RFB are based on the worst case performance, which is just one aspect of the fairness achieved by a packet scheduler. For example, a scheduler that rarely reaches the upper bound of the fairness measure will achieve the same measure of fairness

as another scheduler that frequently or almost always operates at the same upper bound. Therefore, we also need an *instantaneous* measure of fairness that captures the fairness achieved by the scheduler at *any* given instant of time.

To measure the fairness at any instant of time, we also need to consider situations in real applications. The RFB is defined under the assumption that queues are continuously backlogged in the interval of interest. Such an assumption is rarely true in real networks. In networks with real traffic, flow states can change frequently from active state to idle state, or vice versa. However, existing measures of fairness have not taken this factor into account. To effectively guide the design of a fair scheduler, a fairness measure should also be able to capture the performance under the situation where flows change their states unpredictably.

A recently proposed measure of fairness described in [27] addresses these issues. There are two components to this measure: one of how to handle real traffic where flows are not always backlogged and the other of how to measure inequality in service received by the flows. We review these components briefly in this section; a detailed treatment and a more extensive rationale behind the approach may be found in [27].

7.1 Handling a Newly Backlogged Flow

In order to evaluate the fairness of a scheduler in its treatment of a newly backlogged flow, we need to first define the ideally fair way of doing this. We begin with an examination of the ideal but unimplementable GPS scheduling discipline.

Let $B(t)$ represent the set of backlogged flows at time t. Assume that the system starts at time $t = 0$.

Definition 11 *Let $V(t)$ represent the virtual time function [6, 28] (also known as the system potential) at time t. The virtual time is used to track the progress of the GPS scheduler and is computed as follows:*

$$V'(t) = \frac{dV(t)}{dt} = \left(\sum_{i \in B(t)} w_i \right)^{-1} \tag{39}$$

Hence, the service received by a backlogged flow under the GPS server in the time interval $(0, t)$ is given by $w_i V'(t)$. Intuitively, the virtual time represents the ideal fair normalized service that each flow should have received by time t.

Let us now consider a set of n flows served by a scheduling policy P. Consider a case in which the n flows have been backlogged since time t_1. One of these flows, flow i, changes its status from being backlogged to idle at time $t_2 > t_1$ and later

becomes backlogged again at time $t_3 > t_2$. In order to accurately and meaningfully compare the service received by all the flows at time instants after t_3, it is necessary to assign an appropriate value of the normalized service received by flow i until t_3 so that the comparison is over the entire time interval (t_1, t_3). As mentioned earlier, a newly backlogged flow should neither be favored nor be punished for its idle period in the interval (t_2, t_3). Therefore, based on the discussion of the virtual time, the service received by flow i at time t should be $w_i V(t)$. However, if flow i has already received more service than the above amount of service before time t_2 while it was backlogged, then the total assumed service should be $Sent_i(0, t_2)$. This is because a flow that receives excess service should not be able to become idle and then immediately become backlogged again without being disadvantaged later for the excess service it received earlier. These concepts and similar arguments have also been made in [5, 8, 10, 27].

In evaluating the fairness of a specific scheduler it is necessary to keep track of the amount of service allocated by the above method. We borrow the method used in [27] where a per-flow state known as the *session utility* is defined for this purpose. This variable is independent of the scheduling discipline used by the scheduler and is defined as a function of time. Let $u_i(t)$ represent the session utility for flow i at time instant t. Assume that the system starts at time 0. During the period (t_1, t_2) that a flow is continuously backlogged, its session utility is updated as follows:

$$u_i(t_2) = u_i(t_1) + \frac{Sent_i(t_1, t_2)}{w_i} \tag{40}$$

We now discuss how to update the session utility of a flow that just becomes backlogged. Let flow i become newly backlogged or backlogged again at time t. Let $B(t-)$ represent the set of flows that are backlogged just prior to the time that flow i becomes backlogged. Our goal in assigning a session utility value to flow i at time t is to ensure that the comparison between session utilities of all the flows is being made as though the flows have all been backlogged for the same length of time. Accordingly flow i is assigned the following session utility value:

$$u_i(t) = \max\{u_i(t-), V(t)\} \tag{41}$$

With the above definition of the session utility, a newly backlogged flow can be treated as if it had been backlogged for the same length of time as all other flows. Therefore, with a measure which is based on session utility, it is possible to capture the fairness of a scheduler in its treatment of flows that are not always continuously backlogged.

Various measures of inequality have been used in the field of economics for several decades in the study of social wealth distribution and many other economic issues of interest [29]. Some of these methods are related to the theory of majorization used in mathematics as a measure of inequality [30]. This theory has occasionally found use in research in computer networks in the fairness analysis of protocols [31]. The fairness measure proposed in [27] and adopted in this report borrows from a related measure of inequality developed in the field of economics based on the concept of the *Lorenz curve* and *Gini index* [29].

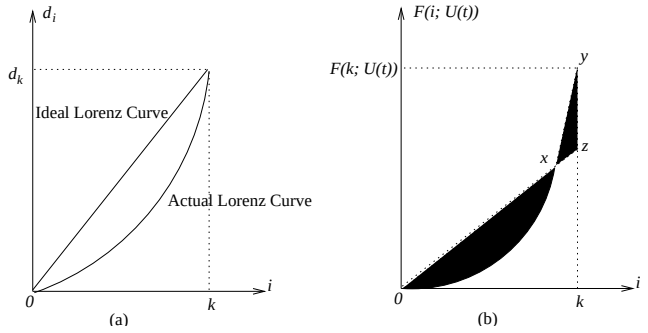

Fig. 10. An illustration of the Lorenz curve and Gini index in the measure of inequalities among (a) income distribution (b) session utilities in a packet scheduler

Consider the problem of measuring the inequality among k quantities, $g_1 \leq g_2 \leq \cdots \leq g_k$. Define $d_0 = 0$, and $d_i = d_{i-1} + g_i$, for $1 \leq i \leq k$. Now, a plot of d_i against i is a concave curve, known as the *Lorenz curve* [32], as shown in Fig. 10(a). Note that if there is perfect equality in these k quantities, the Lorenz curve will be a straight line starting from the origin. The Gini index measures the area between the Lorenz curve and this straight line, and thus measures the inequality amongst the k quantities [29].

In our case, we wish to measure the inequality in the session utilities of the backlogged flows at any given instant of time. The Gini index in our case, therefore, is the area between the Lorenz curve of the actual normalized service received and the Lorenz curve corresponding to the ideally fair GPS scheduler.

When the sum of the k quantities is the same as the sum in the case of perfect equality, the Lorenz curve always lies below the straight line corresponding to the Lorenz curve of the ideal equal case, as shown in Fig. 10(a). However, the sum of the session utilities with a real scheduler is almost never exactly identical to the sum of the session utilities with the ideally fair GPS scheduler. Note that, in a work-conserving scheduler, only the sum of the total service delivered is identical to that in the ideally fair GPS scheduler; the sum of the session utilities is not identical to that in the GPS system. In the Lorenz curve for a work-conserving scheduler, when the sum of the k quantities is not the same as the sum in the case of perfect equality

as with the GPS scheduler, a portion of the curve for the actual scheduler will lie below and another portion will lie above the straight line Lorenz curve for the GPS scheduler. This is illustrated in Fig. 10(b). The sum of the shaded areas in the figure is the Gini index.

The computation of the Gini index is described formally as follows:

Definition 12 *Let $\mathbf{U}(t)$ represent the set of the session utilities of the flows at time instant t when served by a real scheduler and let $\mathbf{G}(t)$ denote a similar set which is obtained when the flows are served with the ideal GPS scheduler. Let $u_{c_1}, u_{c_2}, \ldots, u_{c_k}$ be the elements of the set $\mathbf{U}(t)$, such that $u_{c_1} \leq u_{c_2} \leq \cdots \leq u_{c_k}$. The* Lorenz Curve *of the set of session utilities $\mathbf{U}(t)$ is the function $F(i; \mathbf{U}(t))$, given by,*

$$F(i; \mathbf{U}(t)) = \sum_{j=1}^{i} u_{c_j}, 0 \leq i \leq k$$

The Gini index over the k elements in $\mathbf{U}(t)$ is computed as:

$$\sum_{i=1}^{k} \Big| F(i; \mathbf{U}(t)) - F(i; \mathbf{G}(t)) \Big| \tag{42}$$

As defined above, the closer the Lorenz curve of $\mathbf{U}(t)$ is to the curve of GPS, the smaller the Gini index is, and thus, the fairer the distribution of the session utilities. From the definition of the virtual time function, we know that the normalized service received by each flow at time t in the GPS system is equal to the virtual time, $V(t)$. Hence the Gini index can be computed as :

$$\sum_{i=1}^{k} \Big| F(i; \mathbf{U}(t)) - F(i; V(t)) \Big| \tag{43}$$

With the Gini index, the fairness of a scheduler can be evaluated at each instant during the execution of the scheduler. A comparison of schedulers based on their Gini indices allows us to determine which scheduler achieves better fairness than others at each instant of time.

7.3 Simulation Results

In our simulation experiments, we compare the Gini index of Pre-order DRR with those of other efficient and fair schedulers such as DRR, ERR, Nested-DRR and Pre-order DRR.

In our first set of experiments, the input queues are fed by backbone router traces. We used the traces provided by the National Laboratory for Applied Network Research [33]. Each input is fed by a router trace with a random starting time. Table 2

Source	1	2	3	4	5
Router Abbr. [1]	ANL	APN	BUF	MEM	TXS
Interface	OC3	OC3	OC3	OC3	OC3
$L_{\min}$ (bytes)	28	29	28	32	32
$L_{\max}$ (bytes)	9,180	1,500	1,560	4,470	9,180
r_{avg} (10^6Bps)	0.63	1.4	1.45	0.39	2.1
Weight (w_i)	1.6	3.5	3.7	1	5.5
Link Capacity	6×10^6 Bps				
Total Time	50 seconds				

shows the settings for this set of input traffic. The flow weights are set based on the average rate of each flow. Here we the set the weight of the slowest flow as 1, and weights of other flows are equal to the ratios of their average rate to the smallest rate.

We first extract the length of each packet from the router traces and simulate a scheduling system with continuously backlogged queues. Figure 11 shows the Gini index at periodic instants of time for the schedulers under consideration. Recall that the lower the Gini index, the more fair the algorithm. As is readily seen from the graph, PERR outperforms all the other schedulers.

Table 2: Settings for Traffic Sources from Router Traces

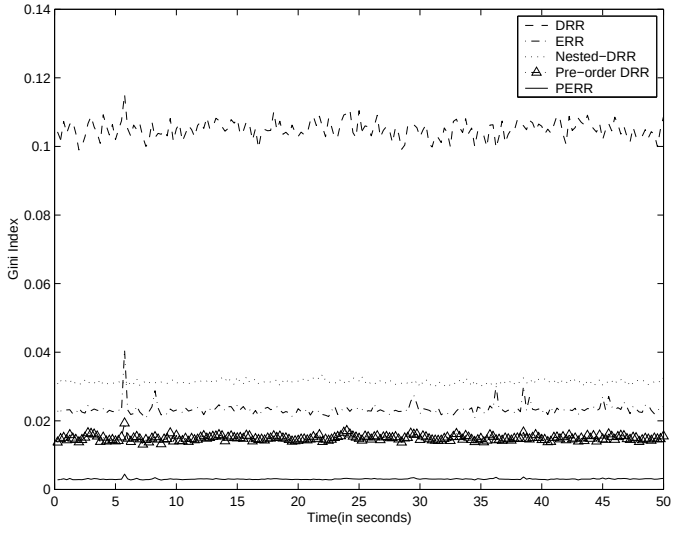

Fig. 11. Gini indices of schedulers on backlogged queues with packet lengths from backbone router traffic traces

[1] The long names of routers are: Argonne National Laboratory(ANL), APAN(APN), University of Buffalo(BUF), University of Memphis(MEM) and Rice University(TXS)

In our next set of experiments, we allow that the flows are not always backlogged, while still using real router traces. Since we are more interested in the performance when the link is close to fully utilized, we set the link capacity such that the sum of average rates of all the flows is 99% of the link capacity. Figure 12(a) shows the average length of arriving packets among all sessions during the simulation interval. Figures 12(b)–(e) show the values of the Gini index observed for the five different schedulers. For the sake of clarity, we plot each scheduler's Gini index on a separate graph, with that of the PERR scheduler plotted on each of the graphs. Once again, we find that the PERR scheduler displays a lower Gini index than any of the other schedulers of equivalent complexity at almost all instants of time.

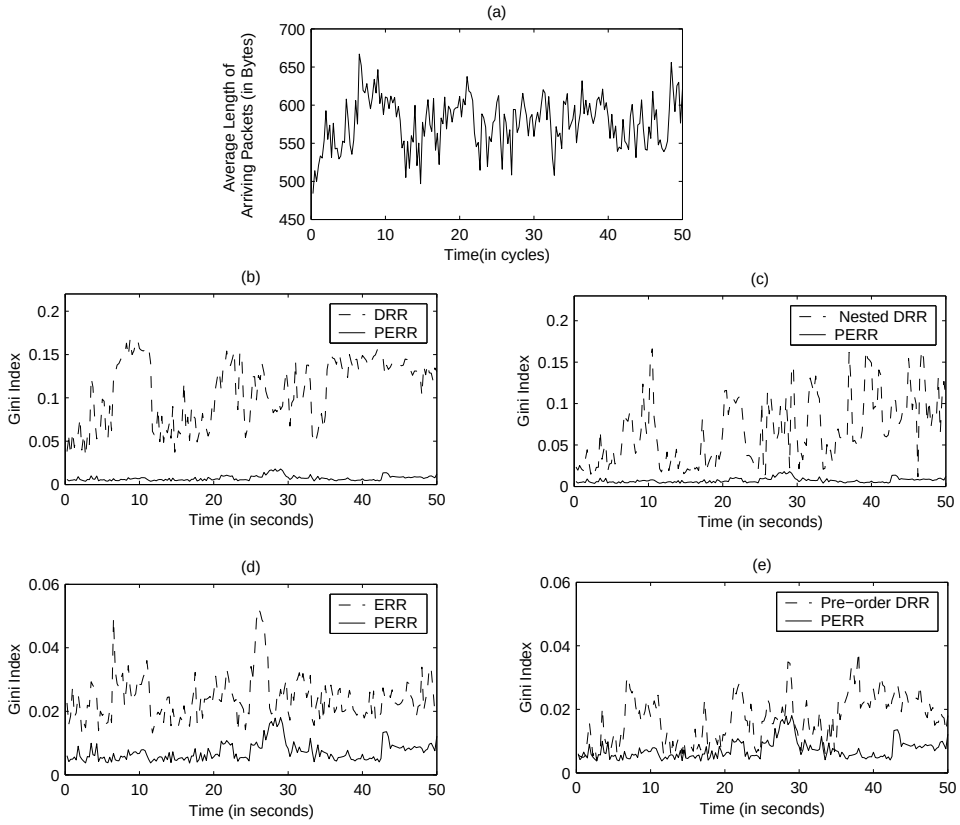

Fig. 12. Gini indices of schedulers with backbone router traffic traces

8 Conclusion

Elastic Round Robin (ERR), a recently proposed fair scheduler, is very efficient with an $O(1)$ work complexity. However ERR is a frame-based scheduler and hence suffers from all the characteristic limitations of these schedulers which arise from the round robin nature of their service order. In this report, we have presented a novel scheduling discipline called *Prioritized Elastic Round Robin* (PERR), which

rearranges the sequence in which packets are transmitted in each round of the ERR scheduler. This is achieved through the addition of a priority queue module consisting of p priority queues. An *Organizer* module dynamically classifies the active flows into these priority queues.

We have analytically shown that PERR has a low work complexity of $O(\log p)$ which is independent of the total number of flows, resulting in a simple and efficient hardware implementation. We also prove that PERR belongs to the class of $\mathcal{LR}$ servers and also evaluate an upper bound on its latency using a novel technique based on interpreting the PERR scheduler as an instance of the Nested Deficit Round Robin algorithm. Our analysis also reveals that PERR has better fairness characteristics and a significantly lower latency bound in comparison to other scheduling disciplines of equivalent work complexity such as DRR, ERR and Pre-order DRR. In addition, we have also presented simulation results, using both synthetic and real traces, which corroborate the conclusions of our analysis.

9 Acknowledgments

The authors are grateful to Ms. Hongyuan Shi for her help in the simulation efforts using the Gini index.

References

[1] S. Keshav, An Engineering Approach to Computer Networks, Addison-Wesley Publishing Company, Reading, MA, 1997.

[2] D. C. Stephens, J. C. R. Bennett, H. Zhang, Implementing scheduling algorithms in high-speed networks, IEEE Journal on Selected Areas in Communications 17 (6) (1999) 1145–1158.

[3] C. S. Inc., Cisco 12016 Gigabit Switch Router: Application Note (1999).

[4] D. Bertsekas, R. Gallager, Data Networks, 2nd Edition, Prentice Hall, Upper Saddle River, NJ, 1991.

[5] D. Stiliadis, A. Verma, Latency-rate servers: A general model for analysis of traffic scheduling algorithms, IEEE Transactions on Networking 6 (3) (1996) 611–624.

[6] A. K. Parekh, R. G. Gallager, A generalized processor sharing approach to flow control—the single node case, in: Proceedings of IEEE INFOCOM, Florence, Italy, 1992, pp. 915–924.

[7] A. Demers, S. Keshav, S. Shenker, Design and analysis of a fair queuing algorithm, in: Proceedings of ACM SIGCOMM, Austin, 1989, pp. 1–12.

[8] S. J. Golestani, A self-clocked fair queuing scheme for broadband applications, in: Proceedings of IEEE INFOCOM, Toronto, Canada, 1994, pp. 636–646.

[9] P. Goyal, H. M. Vin, H. Cheng, Start-time fair queueing: A scheduling algorithm for integrated services packet switching networks, IEEE Transactions on Networking 5 (5) (1997) 690–704.

[10] D. Stiliadis, A. Varma, Efficient fair queuing algorithms for packet-switched networks, IEEE Transactions on Networking 6 (2) (1998) 175–185.

[11] J. C. R. Bennett, H. Zhang, WF^2Q : Worst-case fair weighted fair queueing, in: Proceedings of IEEE INFOCOM, San Francisco, CA, 1996, pp. 120–128.

[12] M. Shreedhar, G. Varghese, Efficient fair queuing using deficit round-robin, IEEE Transactions on Networking 4 (3) (1996) 375–385.

[13] S. Floyd, V. Jacobson, Link-sharing and resource management models for packet networks, IEEE Transactions on Networking 3 (4) (1995) 365–386.

[14] S. Floyd, Notes on class-based-queueing and guaranteed service, Unpublished Notes, `http://www.aciri.org/floyd/cbq.html`.

[15] G. P. H. Adiseshu, G. Varghese, A reliable and scalable striping protocol, in: Proceedings of ACM SIGCOMM, Palo Alto, CA, 1996, pp. 131–141.

[16] S. S. Kanhere, H. Sethu, A. B. Parekh, Fair and efficient packet scheduling using elastic round robin, IEEE Transactions on Parallel and Distributed Systems 13 (3) (2002) 324–326.

[17] S. S. Kanhere, H. Sethu, Fair, efficient and low-latency packet scheduling using nested deficit round robin, in: Proceedings of the IEEE Workshop on High Performance Switching and Routing, Dallas, TX, 2001, pp. 6–10.

[18] S. Tsao, Y. Lin, Pre-order deficit round robin: a new scheduling algorithm for packet-switched networks, Computer Networks 35 (2-3) (2001) 287–305.

[19] S. S. Kanhere, H. Sethu, Low-latency guaranteed-rate scheduling using elastic round robin, Computer Communication 25 (14) (2002) 1315–1322.

[20] Y. Zhou, H. Sethu, On the relationship between absolute and relative fairness bounds, IEEE Communication Letters 6 (1) (2002) 37–39.

[21] K. Thompson, G. J. Miller, R. Wilder, Wide-area Internet traffic patterns and characteristics, IEEE Network 11 (6) (1997) 10–23.

[22] I. Widjaja, A. I. Elwalid, Performance issues in VC-merge capable switches for multiprotocol label switching, IEEE Journal on Selected Areas in Communications 17 (6) (1999) 1178–1189.

[23] D. Stiliadis, Traffic scheduling in packet-switched networks: Analysis, design and implementation, Ph.D. thesis, University of California, Santa Cruz (1996).

[24] S. S. Kanhere, H. Sethu, On the latency bound of deficit round robin, in: Proceedings of the International Conference on Computer Communications and Networks, Miami, FL, 2002, pp. 548–553.

[25] S. S. Kanhere, H. Sethu, On the latency bound of pre-order deficit round robin, in: Proceedings of the IEEE Conference on Local Computer Networks, Tampa, FL, 2002, pp. 508–517.

[26] L. Zhang, Virtual clock: A new traffic control algorithm for packet switching networks, in: Proceedings of ACM SIGCOMM, Philadelphia, PA, 1990, pp. 19–29.

[27] H. Shi, H. Sethu, An evaluation of timestamp-based packet schedulers using a novel measure of instantaneous fairness, in: Proceedings of the Int'l Performance, Computing and Communications Conference, 2003, available at `http://www.ece.drexel.edu/CCL/pubs.html`.

[28] D. Stiliadis, Traffic scheduling in packet-switched networks: Analysis, design and implementation, Ph.D. thesis, University of California, Santa Cruz (1996).

[29] F. A. Cowell, Measuring Inequalities: Techniques for the Social Sciences, John Wiley and Sons, New York, NY, 1977.

[30] A. W. Marshall, I. Olkin, Inequalities: Theory of Majorization and its Applications, Academic Press, New York, NY, 1979.

[31] A. Kumar, J. Kleinberg, Fairness measures for resource allocation, in: Proceedings of the Symposium on Foundations of Computer Science, 2000, pp. 75–78.

[32] J. E. Stiglitz, Economics, W. W. Norton and Co, 1993.

[33] National Laboratory for Applied Network Research, "Passive Measurement and Analysis", `http://pma.nlanr.net/PMA/`.